Development of an application for the generation of robot trajectories based on learning by demonstration techniques

Final Master Thesis

Author:
Dávila Carmona, Cristian

Director:
Casals Gelpi, Alicia
Automatic Control Department

Codirector:
Vinagre Ruiz, Manuel
Automatic Control Department

Master in Artificial Intelligence

Facultad de Informática de Barcelona (FIB)
Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Facultad de Matemàtiques
Universitat de Barcelona (UB)

Escola Tècnica superior d'Enginyeria
Universitat Rovira i Virgili (URV)

January 2017

Abstract

The need of designing and implementing software to prepare robots for the execution of new tasks implies an expensive cost that requires specific hardware, software and knowledge. Learning by demonstration is a paradigm for enabling robots to autonomously perform new tasks learning from previous demonstrations. This project focuses on how to facilitate the learning of new tasks by robots through demonstrations performed by humans. In order to accomplish this goal, this thesis considers two subproblems: *imitation* and *correspondence*. Three different algorithms are used in order to solve both subproblems, in addition to a reinforcement learning to improve the final solution through new demonstrations. Moreover, a methodology is proposed to perform experimentation using these algorithms, including a final discussion over their performance and future work.

Acknowledgements

I would like to thank Alícia Casals for her guide and support in overcoming numerous obstacles I have been facing through this thesis. I am also grateful to Manuel Vinagre for his patience, support, guidance and for helping me getting results of better quality.

Also I would like to thank my friends for assisting me uncountable times to revise the final document and for their infinite patience.Last but not the least. Last, I would like to thank my family for supporting me spiritually throughout writing this thesis and my life in general.

Contents

Abbreviations

DTW: Dynamic Time Warping

GMM: Gaussian mixture model

IK: Inverse Kinematic

ROS: Robot Operating System

Glossary

Baxter: an industrial robot that aims to perform task in a human environment

Dynamic Time Warping: a time-independent algorithm for measuring similarity between two temporal sequences

DTW online joining: An online method to joining demonstrations

DTW hierarchical joining: A batch method to joining demonstrations

End-effector: the actuation device at the end of a robotic arm for the execution of the required tasks

Gaussian mixture model: a probabilistic model for clustering

Inverse kinematics: a technique to compute the joint parameters that provide a desired position of the end-effector

Learning by demonstration: a paradigm for enabling robots to autonomously perform new tasks

Mean path: the mean trajectory generated from a set of trajectories

Chapter 1

Introduction

Usually, in order to prepare a robot to perform a task automatically, it is necessary to program specific software for that task, taking into account different issues (design, implementation, testing, etc). This means that slightly different tasks (like pick and place tasks) will need different software, forcing developers to modify the work done for other tasks. An example of this problem can be a pick and place task for an industrial robot that picks up a door and puts it in a car. If the assembly line requires taking a car bonnet instead of a car door, the developers will need to modify the code to adjust the behaviour of the robot, changing the type of movement, speed, placement, etc. Therefore, new tasks require new specific and functional software, i.e. new development processes, new designs, new tests, etc. Summarising, adding a new task, that is easy to learn from the human point of view, results in additional expensive costs. For a human, changing the task from picking a door to picking a car bonnet is easy, because both tasks are similar, and if the human does not understand how to do it, it is easy to teach him with a demonstration.

Demonstrating how to perform a task is the easiest way humans have to explain tasks; from an example, a human can induce how to perform it[1]. Moreover, for each task execution, a human will improve the execution, learning from errors and focusing on successes[1]. Although it seems unique to humans or animals, learning by demonstration is currently used in robotics[2–6]. In a similar way to humans, a robot can execute a learning task by demonstration. The most straightforward way is to teleoperate[7] the robot while recording all the movements in order to repeat the task in the future. This method is simple, but if the performed action has some errors, the human operator has to repeat the whole task again. A simple solution is to perform different executions of the same task computing a mean of the corresponding trajectories. However, this solution

also has some issues, because the executions may be different in velocity, movement, etc. For humans it is easy to learn quickly from a set of similar examples, discarding the parts that we think are wrong and taking into account the correct ones[1]. In robotics it is not so easy, because it is required to compute how different two examples are, if they belong to the same task, if it is worth to learn from a new example, etc. With a human operator, all these decisions rely on the human, but if it is necessary to automate the process, the mentioned difficulties appear.

Another way of teaching a task by demonstration to a robot is to use force-based learning[7, 8]. This method consists on forcing the robot to perform the task while the operator manually guides the movements. Learning with this method consists on recording either all the movements of all the joints or the path followed by the end effector. In fact, the main difference between learn from a force-based demonstration or a teleoperation demonstration is how the human interacts with the robot to perform the task.

1.1 Motivation

Learning from demonstration has shown to be an effective way to teach robots[2–7] and machines new tasks without the expensive cost of design, implement and deploy specific software for those new tasks. Moving from this specific software to learn from examples allows to perform tasks that are currently costly and inefficient in a more efficiently and accurately way. Moreover, if the software is correctly designed and implemented, a reinforcement learning can be applied[9, 10] in order to improve the execution of the tasks (like humans).

However, for a human is not trivial to perform a task operating the robot or forcing it's movements. This added difficulty can lead to bad or incomplete examples of the task, worsening the robot learning performances. Moreover, usually robots learn only one task from one example, repeating that single example in each execution of the task.

For humans, the most natural way of demonstrating something is to perform the task ourselves. Therefore, in this thesis a system will be designed and implemented in order to learn new task from a human demonstration. The human that perform the task, called demonstrator, will be tracked, recording all the movements performed and storing them for use in later learning. In order to facilitate all the learning process, the system will learn a representation from all the performed examples, returning this representation in the correct structure to be executed by the robot.

The main focus of this thesis, and the main motivation, is to propose a methodology to avoid all the work of implementing code to do new tasks, learning these tasks from demonstrators in the same way that human learn among them.

1.2 Problem

Learning by demonstrations performed by the robot being operated by a human operator avoids many problems[11] like *what to imitate*, *how to recognise the environment* or *how to interact with this environment*. All these problem are solved because the robot is learning from its own movements, i.e. learning from its joints positions, from its own velocity on the execution, etc. When the examples collected come from a human demonstrator, all these problems appear. They appear mainly because of the differences between human and robot structures. Even robotic arms, that usually imitate human arms, present differences that will difficult the correspondence between the movement of the human and the robot arm itself. The first problem, i.e. *what to imitate* problem, refers to what part from the task the robot has to imitate[12]. For example, if a human demonstrator is taking a glass and moving it to another place, what components from this task the robot has to imitate and perform.

Besides the *what to imitate* problem is the *correspondence* problem[12], i.e. *how to recognise and interact with this environment*. The difficulty behind this problem is how to create a correspondence between the demonstrator and the robot, being this correspondence different among all human demonstrators: because humans are different among them, the correct correspondence will vary between demonstrators and robot.

Therefore, the main problem that faces this thesis is **how to facilitate the learning of new tasks by robots through demonstrations performed by humans**, having to solve the *imitation* and the *correspondence* subproblems in order to accomplish the main problem.

1.2.1 *Imitation* subproblem

This first subproblem can be summarised as *in what we look at when it comes to learning a new task*[13]. For humans, this subproblem is solved focusing on the most difficult subtask, trying to understand how to perform it in the same way that the demonstrator does, i.e. *what to imitate*. But in robotics, the robot needs the whole picture, not only the most important or key subtasks. For example, if we pick a cup in order to place it

on a table, the key subtasks are to pick the cup and to place it, not the followed path on the air. Hence, a human will focus the attention on the pick movement and the place movement, obviating the movement through the air. However, for a robot, this obviated movement is also important in order to perform correctly the task.

Moreover, usually human attention is focused only where the action is taking place[1], not in the environment. For example, returning to the pick and place problem, humans will only focus on the hand of the demonstration, being this hand and its movement the answer to the *what to imitate* problem. In robotics, the hand of the previous example is called *end-effector*, and to solve this problem, the robot attention, i.e the examples to learn, will contain only information of this end-effector.

It is trivial to see that focusing on this end-effector solves the problem of *what to imitate*, but it is no the best way. For example, consider the previous example of the pick and place, but adding an obstacle in the middle of the path of the human arm movement (only on the arm movement, not on the path of the hand). This obstacle could be a jar of water contiguous to the elbow of the demonstrator, and in order to avoid this jar, the demonstrator will move the whole arm over the jar. If the robots only focus on the end-effector, at the time of executing the task, it will throw the jar instead of avoiding it.

Therefore, focusing only in the end-effector will lead the robot to the next problem: the **correspondence** subproblem.

1.2.2 *Correspondence* subproblem

One of the main reasons that allows humans to learn from other humans is that they share the same environment, the same perception of their environment and a similar body, facilitating the correspondence of the movement between demonstrator and learner. Except in special cases, humans will do visual demonstrations in the same environment (or at least pretty similar), facilitating that the learner can focus on the task without having to worry about how the demonstrator perceives the environment. Moreover, for humans the visual perception and the representation of the environment is the same. For example, returning again to the pick and place task, for both human demonstrator and human learner, the table is perceived in the same way and it means the same thing: a table. But for a robotic learner, the table is not perceived in the same way, and does not mean the same thing. Moreover, it is possible that the robot does not know that the table exists, because the table has not been mapped in its memory:

the table is not inside of the *what to imitate* subproblem, therefore it is not necessary to give that information to the robot.

Moreover, the correspondence of bodies between a human demonstrator and a human learner is performed automatically by their brains, but in the case of a robot, this correspondence is not trivial. Considering a robotic arm with three joints (shoulder elbow and wrist), the correspondence between this robotic arm and a human arm is straightforward: each joint of the robotic arm correspond with a joint of the human arm. But many robotic arms have more joints or the joints are placed on different points that do not match the ones from a human arm, making this direct correspondence between joints impossible.

In summary, to solve the correspondence problem[13] takes more than giving computer vision and object recognition to the robot, because a mapping between the human interaction to the robotic interaction is needed.

1.2.3 *Improving* the knowledge learned

As discussed above, human are continuously learning from the same task in each execution, improving their current knowledge. Without this current learning, the knowledge acquired will be constant, without further improvements. Therefore, without adding a way of relearning or reinforce the learning, the final solution will be fixed and will not be possible to improve the execution of a task with new demonstrations. Hence, this fixed knowledge appears in the previous imitation and correspondence subproblems, and that is why it is faced at the same time that them.

1.3 Proposed solution

In order to solve both subproblems in conjunction with how to *improve* the knowledge learned, and therefore to solve **how to facilitate the learning of new tasks by robots through demonstrations performed by humans**, this thesis develop a first solution for the first subproblem, expanding it to solve the second subproblem and applying on both of them a reinforcement learning algorithm to improve the knowledge learned, being three the algorithms used in order to solve both subproblems.

1.3.1 Solution for the *imitation* subproblem

As explained before, *what to imitate* is a problem that can be first solved tracking the end-effector of the demonstrator. In the case of the pick and place example with a cup performed why a human, the end-effector could be the hand of the demonstrator. Therefore, it is needed to record all the movement of this end-effector to track the followed trajectory.

Therefore, the solution proposed on this thesis for this subproblem is to track the path performed by the end-effector in each demonstration of the task, compute a *mean path* from all the demonstrations in the same reference frame and to transform this path to the robots' joints using inverse kinematics.

However, the main problem of this solution is that all the information about the position of the whole arm is lost, leaving the robot with all the computation of the joints. Because the robot does not have information about the environment, it is impossible for it to compute a solution avoiding obstacles, leading to possible conflicts on the real environment with other objects. This problem is the one that leads to expand this solution in order to solve the correspondence subproblem.

1.3.2 Solution for the *correspondence* subproblem

A way to solve this subproblem is to create an exact map between the demonstrator arm and the robotic arm. However, an exact mapping is only possible with a direct correspondence between joints, and as explained before, this direct correspondence does not happen with all the robotic arms. Because an exact mapping cannot be created, a mapping good enough is needed to simulate the same movement between both arms (demonstrator and robot). Without being able to apply the same joints' positions to all the joints from both arms, the best solution is to keep the same orientation for the end-effector and to perform the movement in an anthropomorphic way.

In order to keep the same orientation, it is need to track two joints from the demonstrators' arm: the end-effector and the second joint. For example, in a human arm the end-effector is the hand and the second joint is the elbow. Therefore, the solution proposed on this thesis in order to solve this subproblem is to track both hand and elbow in each demonstration, computing the mean path from the demonstrators' end-effector and the orientation from both joints. In addition, the inverse kinematic will be computed keeping the arm in an anthropomorphic way.

1.3.3 Solution for how to improve the knowledge learned

Every time a human performs a task, this execution is used to reinforce the task learned, improving it increasingly. This reinforcing process can affect to different but similar tasks in a positive way, improving the execution of these tasks without doing them. This thesis does not aim to imitate that advanced kind of reinforcement learning, but to implement a similar one that rates a new example of an specific task and chooses between learning from it or discarding it. When a human demonstrator performs a new demonstration, this new demonstration will be processed by the system, computing how different it is from the current task. If the new demonstration is different (the demonstration is bad or the user provides another task), the system will discard it, but if the new demonstration is similar to the current task, the system will learn from it. Using this reinforcement learning avoids to use bad examples and helps to improve the execution of the current task. Moreover, the system will keep different tasks on memory, reinforcing the selected one while keeping available all the other tasks. Therefore, this solution is applied in conjunction with the previous solutions for their corresponding subproblems.

1.4 Objectives

In order to organise, structure and execute the whole project, it is a must to establish the main objectives and subobjectives to solve the problem of *how to facilitate the learning of new tasks by robots through demonstrations performed by humans.*

As explained in section 2, the main problem is divided in two main subproblems: the *what to imitate* and the *correspondence* subproblems. Since the objectives of each subproblem are in fact the same, the same methodology and objectives are followed.

The subobjectives for both main subproblems are the following:

- Data extraction and preprocessing
- Choose the algorithm to execute
- Apply reinforcement learning
- Evaluation of the solution

The next subsections will explain the methodology per each subobjective.

1.4.1 Data extraction and preprocessing

In order to learn from a human demonstrator, all the movements have to be captured using Polhemus Fastrak (tracking hand and elbow) and formated correctly to be computed by the system. The following tasks are followed to accomplish this subobjective:

- Install all the Polhemus software
- Perform a movement and store all the information on a file with the correct structure
- Preprocess the data stored on the file

The first and second tasks are pretty related, because the second task is used to verify the first one. The third task can be performed directly on the storing part of the second one, because it consists on saving all the information with the correct format, order and structure.

1.4.2 Learning Algorithm

This subobjective is applied for each of the three algorithms that will be explained in Chapter 2, being the tasks the following:

- Analyse the data obtained by the tracker (volume, type, structure, format)
- Implement the algorithm
- Test the implementation

The first task is required in order to implement all the algorithms depending on the data type (int, double, float, etc) and amount of data (coordinates, angles, etc).

1.4.3 Adding reinforcement learning

This subobjective consists in implementing the reinforcement learning with the algorithms chosen in the previous subobjective. The required tasks are:

- Choose the policies for the reinforcement learning algorithm
- Implementation of policies and the algorithm
- Test the implementation

1.4.4 Strategy review, evaluation and control

In order to evaluate the performance of the system, some metrics are applied through different tasks:

- Choose the right metrics for evaluation and control
- Apply the evaluation metrics
- Test the system
- Execute the system with a robot (a two arms Baxter robot)

1.5 Limitations

On a thesis with a project so big and easy to extend a risk appears to spend resources in too many aspects without significantly advance in none of them. Therefore, it is a must to establish limitations and to specify the scope of the project. For the *imitation* subproblem, the scope of the thesis is to track the end-effector (i.e. the hand) of the demonstrator, compute the mean path from scratch or with the reinforcement learning and to perform this mean path with the robots' hand using the whole arm (not only moving the joints of the hand). It is not inside the scope of the project to build a system in order to move in real time the robots' arm according to the demonstrators' arm neither to map the whole environment to the robot. For the *correspondence* subproblem, in addition to the previous limitations, only the mean path for the end-effector and the orientation for the elbow will be computed.

The mapping computed on the *correspondence* subproblem will not be the *best* mapping, but one good enough to be an acceptable solution. It is not inside the scope to learn how to avoid objects in the environment, but to learn the mean path that avoids those objects. It is inside how the end-effector moves, but not the movement itself: if the end-effector is a robotic hand, it is not inside the scope the movement of the robotic fingers, neither how the hand grabs an object.

1.6 Document structure

Chapter 1 introduces the topic of Learning by demonstration, the problem to face, how to solve it and the limitations of the thesis. Chapter 2 provides all the theoretical framework

in order to understand how this thesis faces the main problem and its subproblems. Chapter 3 gives an overview of how the system works, focusing on the different technical parts. Chapter 4 explains the proposed experimentation in order to perform all the experiments correctly. All the experimentation is explained in Chapter 5, including a discussion per each experiment and result.

Chapter 2

Theoretical Framework

This chapter aims to provide concepts and information which are necessary to understand how this thesis processes and compute the acquired data from a action or trajectory demonstration to generate and send the correct joints positions to the robot.

2.1 Theoretical Framework

This section will provide the theoretical framework and the concepts needed to understand the different implemented algorithms and how the system works and processes all the demonstrations to compute the correct mean path. It is important to point out that the three chosen algorithms are the geometric mean, Gaussian mixture models and Dynamic time warping. These algorithms are the most commonly used because they are computationally fast and have proven their effectiveness in a multitude of projects[14–20].

2.1.1 Mean path

When learning by demonstration is applied with different demonstrations of a task, the aim is to learn a correct representation from these demonstrations. For example, if these demonstrations are repetitions of a pick and place exercise, the goal will be the correct way of doing the pick and place actions. As stated before, the aim of this project is to learn a correct path for the end-effector from different demonstrations of the same task (e.g. pick and place), without focusing on pick or release movements. Therefore, the problem consists on learning a mean path from the demonstrations and to evaluate how correct is this path.

Before facing the problem of *how correct* a solution is, must have been decided *what* a solution is. In the scope of this project, the mean path learned from different demonstrations is the solution. This mean path will consist of a set of coordinates with their corresponding times. For example, if the demonstrations are from a task of picking a mug from a table and placing it on the other end of the table, the mean path will be a path that starts where the mug is and finish where the mug has been placed, having in each point of the path the time when the end-effector has to reach it.

It is trivial to see that the start and end points of all demonstrations have to be the same, or else the first and last points on the mean path will vary. A simple normalisation of the coordinates on the space can be applied. However, with this normalisation, all the information about the movement itself is lost, because it is impossible to know if the movement starts on the table or one metre above. Moreover, if the coordinates are normalised, it is necessary to indicate the robot where to start the movement. However, without normalising the coordinates, the movement will start on the starting point learned, being inside the reference space of the robot.

In addition to normalising or not, it is important to understand that the space of all the demonstrations will be in the same space than that of the robot, because they are in the reference axis of the demonstrator, and therefore, the same space than the robot. Learning from one space and applying it into another space is out of the scope of this project.

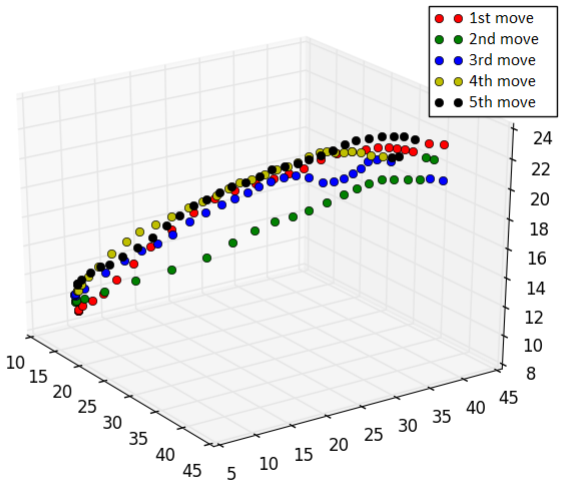


FIGURE 2.1: Example of paths from a diagonal movement.

Figure 2.1 shows a simple set of demonstrations from a straight line movement. As can be seen, not all the demonstrations start or finish on the same coordinates, nor the

movement itself is exactly the same. This example lead us to another issue of the mean path: how similar is this mean path with that of the demonstrations. For example, if some demonstrations have a movement in serpentine and others in straight line, what is the correct mean path, the straight line or the serpentine? This problem is the same that was commented before about *how correct* a solution is. The solution for this problem is discussed in section 4, which has all the information about the evaluation metrics.

Nevertheless, it is important to clarify that demonstrations similar to each other, i.e. belong to the same movement. Therefore, if the demonstrations belong to the same movement but they differ on some aspects (like the straight or serpentine line), it means that these differences are not important, and for the mean path it is not important whether it presents a serpentine or a straight line. Furthermore, if we think in a subset of demonstrations with a straight line and another subset with demonstrations with a serpentine line, the mean path will look more like the bigger subset.

As explained before, the demonstrations consist in a movement in a 3D space. For each demonstration, the number of coordinates is known, and therefore it is simple to create a new vector with the normalised instant of time in which the coordinate was captured. For example, if we have a demonstration with 6 coordinates (size is 6), it is known that at instant 0, the first coordinate was captured, at the instant 1 the second coordinate was captured and so on. Normalise this temporal vector consists in normalising it in a range between 0 and 100. Therefore, all the demonstrations with this normalised temporal vector allow to know in which instant each coordinate was captured.

2.1.2 Geometric mean path

The first way to create a mean path from the demonstrations is to apply a geometric mean between the points of the demonstrations. This solution consist in sorting the points of the demonstrations, dividing these points into bins and computing the geometric mean of these bins. The result will be a set of mean coordinates sorted by time, i.e. the first coordinate on the set will be the starting coordinate of the path, and the last coordinate on the set will be the end of the path. Figure 2.2 shows an example of these bins with the previous example.

The first step, sorting the points, consists in sorting all the points of the demonstrations in only one vector of coordinates. This sorting is performed according to the instant of time the coordinate belongs to. As explained before, each coordinate has the normalised instant of time in which it was captured. At the end of this step, the vector will have

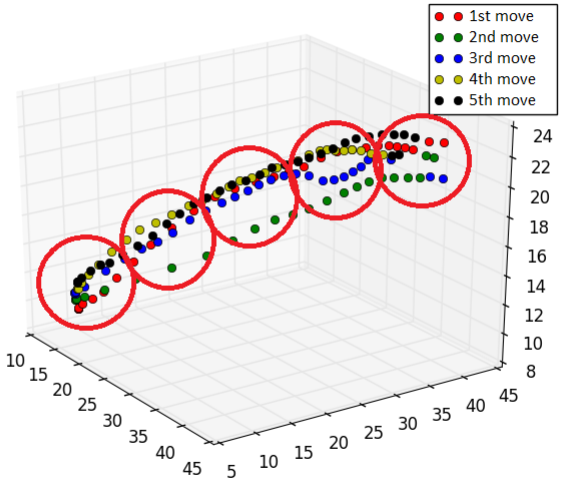


FIGURE 2.2: Example of 5 geometric bins.

all the coordinates from the demonstrations sorted, simplifying the creation of the mean coordinate per each bin of coordinates.

The second step will divide the sorted vector of coordinates into bins. Given a number of bins k, the vector will be split into k slices of the same size. Thus, in this way it is possible to group coordinates with a similar temporal instant in a bin. It is straightforward to see that the value of k will set the granularity of the solution, i.e. how well the mean path will adjust to the demonstrations. Therefore, with bigger values of k, the mean path will tend to overfit, and on the contrary, with small values of k the mean path will tend to underfit.

The last step consist on applying a geometric mean on each bin. The geometric mean coordinate per bin will be a coordinate on the mean path. Whereas the coordinates on each bin are sorted by the temporal instant in which they were captured, all the bins are temporally sorted. Hence, the mean path will also be sorted.

This solution is simple and computationally cheap, because only two operations are performed: sorting and computing the geometric mean. But it also has some difficult issues that do not make it feasible for this project. First, the solution will vary significantly according to the value of k. This dependence does not allow to automatise all the process of computing the mean path in a simple way, because for tuning k with different values, and therefore, choosing between different solutions will add complexity to the system while requiring a classifier to establish if the mean path has to be overfitting or underfitting according to the type of exercise. For example, in a pick and place exercise,

where the general path followed is important, the mean path has to be underfitting, but in a writing exercise, the mean path has to be overfitting in order to learn the correct handwriting performed.

Moreover, this solution prevents to correctly compute the key coordinates on the exercise. For example, going back to the pick and place exercise, the starting and ending points are the most important points on the demonstrations. With this solution, these points will be diffuse.

2.1.3 Gaussian mixture models

A mixture model is a probabilistic model that represents the probability distribution of observations in the overall population. The model assumes that all the data points are generated from a mixture of a finite number of distributions with unknown parameters. Therefore, a Gaussian mixture model (GMM)[21] will assume that this finite number of distributions are Gaussian distributions. A way of understanding the Gaussian mixture model is to see it like a generalised k-means clustering, where, in addition to create the clusters from the centres of the latent Gaussians, the model incorporates information about the covariance structure of the data.

A mixture model can be seen as a hierarchical model with the following structure:

- N number of observations
- K number of mixture observations
- A set of K parameters, each specifying the parameter of the corresponding mixture component.
- A set of K mixture weights (each of which is a probability, all of which sum 1).
- K corresponding random latent variables specifying the identity of the mixture component of each observation

In fact, the Gaussian mixture model used in the project uses the expectation-maximisation (EM) algorithm for fitting mixture-of-Gaussian models. This is the fastest way for learning mixture models, but as this algorithm maximises only the likelihood, it will not bias the means towards zero, or bias the cluster to have specific sizes that might or might not apply. Therefore, in some occasions the mean path resulting (and depending on the type of exercise) will look like the demonstrations but doing some kind of zigzag. Moreover,

as a clustering algorithm, GMM will need a number of clusters K. Thus this algorithm has the same problem as the geometric mean algorithm, because the value of K will condition the mean path, and in the same way as the geometric mean, this solution prevents the correct learning of the important coordinates of the task (like the start and end coordinates). Furthermore, the algorithm fed with both spatial coordinates and time, will lead to increase the zigzag movement (because some clusters will have spatial coordinates that do not agree with their instant of time).

2.1.4 Dynamic time warping

Dynamic time warping (DTW) is an algorithm for measuring similarity between two temporal series[16–20] with time independence. This independence over time guarantees that DTW will found a correspondence between two temporal series. DTW has been used in different applications like speech recognition, speaker recognition, video series, audio, graphics data, etc. Actually, any data which can be turned into a linear sequence can be processed by DTW.

The main use of DTW is to compute the distance between two series. This algorithm will compute how different two series are, independently of the length of the sequences. For instance, given two sequences, X and Y, being Y a copy of X with duplications, DTW will return a distance of 0, i.e. both series are equal.

Briefly, DTW will try to align two sequences computing their optimal matching. This matching is performed over the elements of the sequences: the sequences are *warped* in the time dimension matching the elements with more similarity.

Following the previous example, lets set two time series X and Y, being both positions over a 1-dimensional space.

$X = [1, 2, 3, 4, 5, 6]$
$Y = [1, 1, 2, 2, 3, 4, 4, 5, 6]$

It can be said that X is fastest than Y because in each step X moves 1 position while Y needs 2 steps. In this example, DTW will be used to measure the similarity of both series (i.e. the paths) using the Euclidean distance. If the speed is measured, it is trivial to see that they are not equal, but if the similarity of the paths is measured, can be seen that both are equal. A human will reach this conclusion comparing each position in the series and looking for the matching between them. In a similar way DTW will work in order to find the similarity. At the end of the execution, the algorithm will return the difference between both series, being in this case 0 (they are equal).

Internally, DTW will compute a matrix of distances between the elements of both series. Each position on the computed matrix is the cost to reach that element from the initial position. This distance matrix is used to compute the DTW path, i.e. the path that matches both series.

Following the previous example, on table 2.1 the starting distance matrix can be seen. The initial position (down left on the matrix, at position (x=0, y=0)) has a cost 0 because both series start on the same position.

6						
5						
4						
4						
3						
2						
2						
1						
1	0					
	1	2	3	4	5	6

TABLE 2.1: Distance matrix initialisation

On table 2.2 the first iteration of the DTW algorithm can be seen. Now, besides the first position, there are the costs for positions (1,0), (1,1) and (0,1). Taking a look over the position (1,0), it can be seen that the cost is 1, but for position (0,1) the cost is 0. This cost is obtained because the distance from 1 to 2 is 1, but from 1 to 1 is 0.

6						
5						
4						
4						
3						
2						
2						
1	0	1				
1	0	1				
	1	2	3	4	5	6

TABLE 2.2: Distance matrix on the first step

On table 2.3 are the costs computed on the second step of the DTW. It can be seen that some costs are increasing (on the matrix positions (3,0) and (3,1)) while in the matrix position (2,3) the cost is 0.

6						
5						
4						
4						
3						
2						
2	1	0	1			
1	0	1	2			
1	0	1	2			
	1	2	3	4	5	6

TABLE 2.3: Distance matrix on the second step

On table 2.4 the completed distance matrix can be seen. It is trivial to see that in the matrix there is a path of zeros between the matrix positions (0,0) and (6,9): the DTW path. This path is used to compute the similarity between both series because it is the

path that joins both series. For this example, it can be seen that both series are equal, because all the elements (cost) on the DTW path are zeros. Moreover, this is a simple example to see how the DTW is independent of both duplications and time.

6	5	4	3	2	1	**0**
5	4	3	2	1	**0**	1
4	3	2	1	**0**	1	2
4	3	2	1	**0**	1	2
3	2	1	**0**	1	2	3
2	1	**0**	1	2	3	4
2	1	**0**	1	2	3	4
1	**0**	1	2	3	4	5
1	**0**	1	2	3	4	5
	1	2	3	4	5	6

TABLE 2.4: Distance matrix completed

Therefore, this path can be seen like:

DTW path = [(0,0),(0,1),(1,2),(1,3),(2,4),(3,5),(3,6),(4,7),(5,8)]

where each element is a position on the distance matrix.

Algorithm 1 is the pseudo-code for the DTW algorithm. Taking a look at the inner loops, it can be deduced that the whole execution of the algorithm has a cost of NxM, where N is the size of the first series and M the size of the second, but generally the cost of the algorithm is $O(N^2)$.

Data: $s : array[1..n]$, $t : array[1..m]$

Result: Similarity

$DTW \leftarrow array[0..n, 0..m]$;

for $i \leftarrow array[0..n]$ **do**
  $DTW[i, 0] \leftarrow infinity$

end

for $i \leftarrow array[0..m]$ **do**
  $DTW[0, i] \leftarrow infinity$

end

for $i \leftarrow array[0..n]$ **do**
  for $j \leftarrow array[0..n]$ **do**
    $cost \leftarrow d(s[i], t[j])$;
    $DTW[i, j] \leftarrow cost + minimum(DTW[i-1, j], DTW[i, j-1], DTW[i-1, j-1])$;
  end

end

return DTW[n, m]

Algorithm 1: Dynamic time warping algorithm

Algorithm 1 returns the similarity of both series, but it is easy to add a few lines of code to return the DTW path (a simple BFS algorithm).

2.1.4.1 Join DTW paths

As explained before, the execution of the DTW with two temporal series will return the similarity and the DTW path. The main use of this path (in addition to computing the similarity) is to create a join of both temporal series. As can be seen in the previous example, the computed DTW path is a vector of positions in the distance matrix:

DTW path = [(0,0),(0,1),(1,2),(1,3),(2,4),(3,5),(3,6),(4,7),(5,8)]

Each element of the path marks the correlation of pairs of elements on both temporal series. For example, the first element of the path is (0,0), meaning that the first element of the first temporal series is related to the first element of the second temporal series. For the second element of the path, that is (0,1), it means that the first element of the first temporal series is related to the second element of the second temporal series. It is very useful to know this correlation, because computing a join path between temporal series is now trivial. In order to compute this join path, it is only necessary to compute the mean of the related elements between both temporal series (table 2.5):

DTW path	First serie	Second serie	Mean value	Merged series
(0,0)	1	1	(1+1)/2=1	**1**
(0,1)	1	1	(1+1)/2=1	**1**
(1,2)	2	2	(2+2)/2=2	**2**
(1,3)	2	2	(2+2)/2=2	**2**
(2,4)	3	3	(3+3)/2=3	**3**
(3,5)	4	4	(4+4)/2=4	**4**
(3,6)	4	4	(4+4)/2=4	**4**
(4,7)	5	5	(5+5)/2=5	**5**
(5,8)	6	6	(6+6)/2=6	**5**

TABLE 2.5: Table computing merged series

Therefore, in order to compute a mean path, all demonstrations should be joined in pairs. At this point, two different ways of performing this computation appears: online or in batch. The first way consists in computing this joined path between the first demonstration and the second, joining the third with the resulting join of the first and the second, then joining this new path with the forth and so on (as can be seen in figure 2.3). The main problem of this solution is that in the final mean path computed, not all the demonstrations have the same importance, because the last demonstration will have more importance than the first one: the last will be joined just once, while the first one will be joined n-1 times (where n is the number of demonstrations).

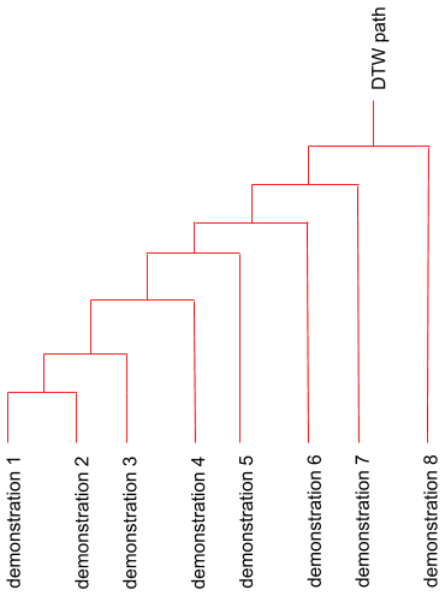


FIGURE 2.3: Example of joining in an online way.

This handicap leads to compute the mean path in batches of demonstrations, being these unions of hierarchical form (as can be seen in figure 2.4), avoiding that some demonstrations have more importance on the mean path than others. In spite of if the number of demonstrations is not a power of two some demonstrations will be more important than others, it is avoided the huge difference done with the online solution. Therefore, in the case that the number of demonstrations is not a power of two, the joining will follow a left association.

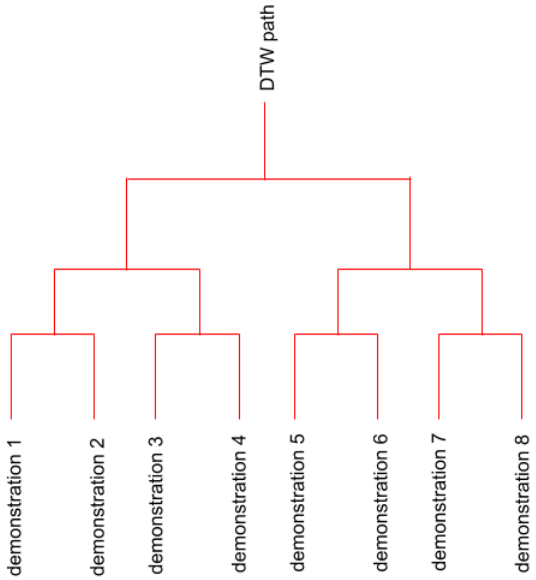


FIGURE 2.4: Example of joining in a hierarchical way.

2.2 Reinforcement Learning

Reinforcement learning aims to improve the knowledge acquired taking actions in order to reinforce this knowledge. In this thesis, reinforcement learning is used to reinforce the mean path with new demonstrations of a task.

In order to avoid misleading demonstrations or demonstrations from another task, a learning policy is used. This policy aims to decide whether it is worth learning from a given demonstration or it is best to discard it. In this project two policies will be used: the DTW distance between the mean path and the new demonstration; and the users' opinion. The first policy will be the default policy, but the implemented system will allow users to decide if they really want to use that new demonstration.

In order to apply reinforcement learning, a training set is needed to be used as knowledge base. This training set is a set of demonstrations to compute the first mean path computed (the base). When a new demonstration is given, the system will compute the DTW distance between this new demonstration and the mean path, and if this distance is less than a predefined threshold, the demonstration will be added to the training set and a new mean path will be computed. This threshold prevents unwanted demonstrations from being added to the training set.

Therefore, on the reinforcement learning framework, the state **s** on a instant of time **t** is that mean path, the actions **a** are to accept or not the new demonstration, and the evaluation **e** is the evaluation metric.

2.3 Human-Robot mapping

In order to move the robotic arm in the same way as the human arm does, a mapping between the human arm and the robotic arm is needed[22]. For simplicity, two solutions are considered, being both solutions the most common solutions in the literature about this topic. In fact, in this thesis each solution is applied only to one of these two subproblems: *imitation* subproblem and *correspondence* subproblem.

The solution for the mapping in the *imitation* subproblem is exactly the solution itself: to perform a Forward/Inverse Kinematics approach. This approach consist in tracking the final actuator of the demonstrator and imitating it by the robotic arm. Therefore, applying the solution explained before for the *imitation* subproblem will solve the subproblem itself and the mapping between the demonstrator arm and the robotic arm.

This solution is valid because all the work is performed by the robot in the computation of the inverse kinematic, and is not restricting angles and rotations for the different joins of the robotic arm.

The solution for the mapping in the *correspondence* subproblem is more difficult than the previous one, because in this solution it is needed to restrict possible movements of the robotic arm in order to imitate the movement of the human arm, not only the final actuator. Because a direct mapping is not possible, a simple solution is to compute the orientation for the end-effector, keeping the same orientation than the human demonstrator hand. With this restriction, and due the similarity between the human arm and the robotic arm, when the inverse kinematic is performed the robotic arm will have a similar movement to the human arm. Therefore, this solution will add the mentioned orientation on the computation of the inverse kinematics in order to restrict the possible movements of the robotic arm to imitate the position of the human arm.

2.4 Inverse Kinematics

In robotics, Inverse Kinematics[23, 24] (IK) is the technique that allows to determine the joint parameters that provide a desired position of the end-effector for the robot. Given a coordinate and a orientation, IK aims to find the values for the robots' joints in order to position the end-effector in the desired spatial localisation, depending this values on the robot configuration. Furthermore, per each position and orientation, there may be multiple solutions.

2.5 Evaluation metrics

One of the pillars of this thesis is the evaluation of the mean paths obtained in order to evaluate and choose which algorithm performs better. The main issue with how to evaluate the mean path is that the correct path to perform a task is not know, therefore it is not possible to compute whether a solution is 20% correct or 90% correct. In order to choose a metric, how the demonstrations are related between them has to been considered. To perform a mean path means that this path is the mean of all the demonstrations, therefore the mean distance between the ideal mean path and the mean distance between demonstrations have to be the same. For instance, imagine a set of 8 demonstrations, with a mean distance between them of 10. This distance means that the mean distance (difference) between them is 10, therefore the ideal mean path

will have a mean distance (difference) of 10 to the demonstrations. Then, evaluating a mean path is needed to compute the mean distance between this mean path and all the demonstrations, in order to compare this mean distance with the mean distance between demonstrations. The closer this distances are between them, the better the solution. In the ideal case where the demonstrations are exactly the same, the distance between them will be 0, having the ideal mean path a distance of 0 between them.

However, in the real world it is almost impossible to have this distance of 0, and since there is not a maximum distance between demonstrations, to evaluate the mean path only this two distances can be compared between them. If the distance between demonstrations is small, it is easier to compute a correct mean path, because the variance between demonstrations will be small, leading to a better solution. But if the distance between demonstrations is large, the variance will be also large, possibly leading to a worse mean path.

How to evaluate the orientation can be performed checking the orientations in each point for each demonstration and comparing it to the orientation in each point of the mean path. However, this method can not evaluate how anthropomorphic the robots' arm is when perform the mean path. Therefore, the best way to evaluate how affect the orientation is with a visual evaluation when the robot is performing a task.

Chapter 3

Technical Framework

This section aims to provide all the technical background needed to understand and recreate this thesis, giving in addition an overview of the whole system.

3.1 System overview

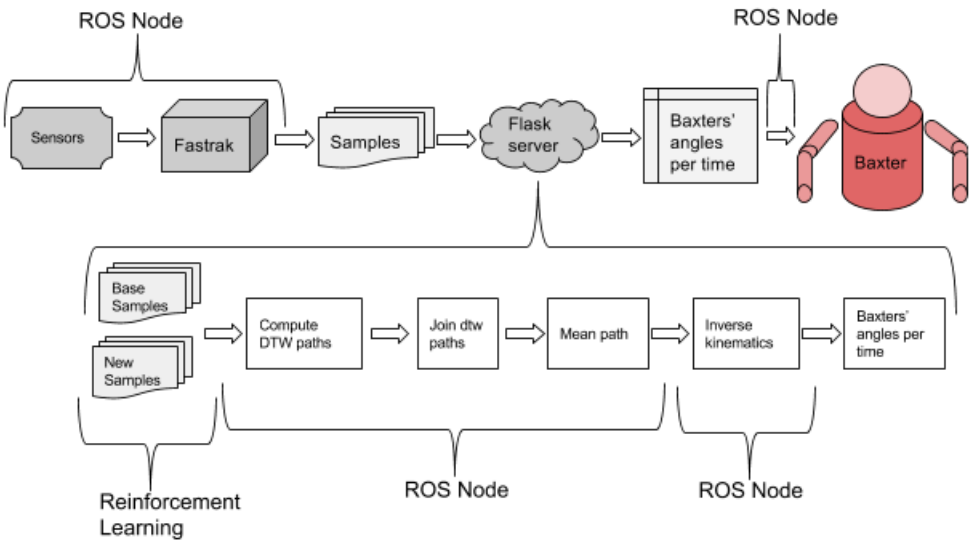


FIGURE 3.1: System overview.

Figure 3.1 shows the link between the three technologies used in this thesis. This system allows to track a human arm, compute the mean path and send the correct instructions to the robot in order to perform that path. As will be explained in the following sections, the demonstrator will have two sensors placed on his arm: one on the hand and another on the elbow (only for the *correspondence* subproblem). The tracker will capture the coordinates of each sensor, sending all this information to a ROS node to save it on a file. This file, that contains all the information of the demonstration, will be uploaded to a

local server to compute the mean path (and to ingest it on the reinforcement learning). When the learning is performed, the server will return the computed path and the user can send it to the robot using a ROS node.

The path that returns the server is in joints positions and time, and it can be sent through a ROS node that communicates with the robot. All the ROS nodes are independent of the number of sensors that the tracker has, therefore the same configuration is used to face both subproblems.

3.2 Baxter

Baxter is a robot manufactured by Rethink Robotics[25] that aims to perform tasks in a human environment sharing the same tools than humans. Baxter aims to perform the most repetitive and non-key task on assembly lines, avoiding to use human workers that can make mistakes and get tired easily, while reducing production costs in those tasks. Baxter is safe to operate in production environments and can be manually trained by operators (force-based learning), reducing the time and cost of third party programmers. Figure 3.2 shows Baxter in a production environment, replacing a human worker on assembly line.

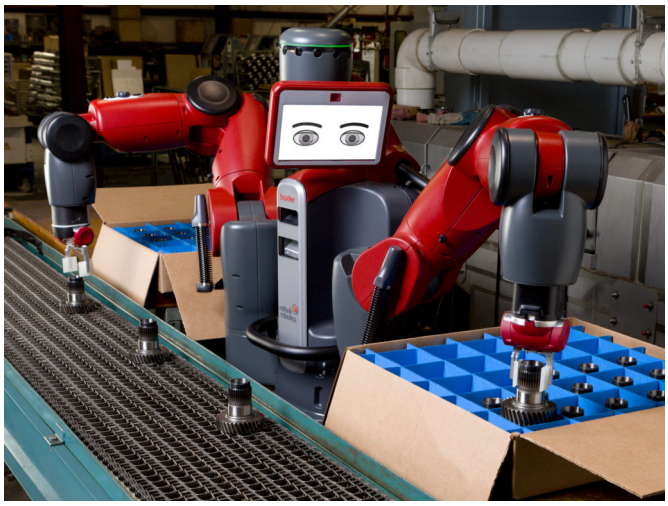


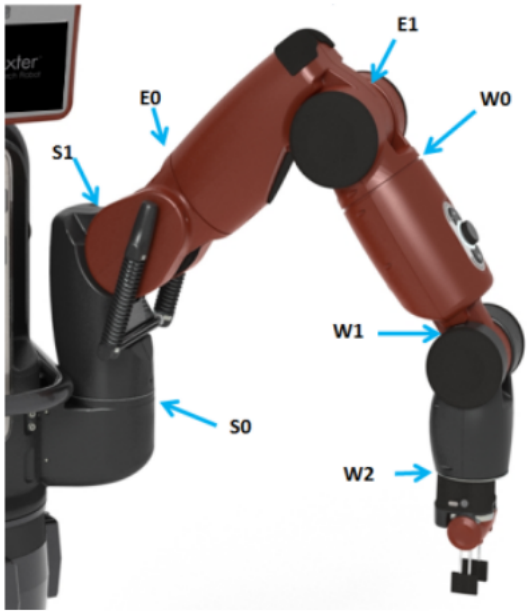
FIGURE 3.2: Baxter in a production enviorement.

3.2.1 Technical specifications

Baxters' body[26] is composed of two different structures: the robot itself and the pedestal that supports the robot. Including the pedestal, the height is between 177'5 cm and 190'5 cm (the pedestal is adjustable). The body weight is of 75 Kg without

pedestal and 138 Kg with pedestal. Each arm has 7 degrees of freedom (14 in total), has a maximum reach of 1210 mm and including the end-effector, each arm can support a maximum payload of 2'2 Kg. The robot is powered by a Intel Core i7-3770 Processor, has 4Gb of RAM memory and a 128 Gb SSD. Baxter has a display on the head with a resolution of 1024 x 600 pixels that is used to interact with users and operators. Moreover, it has a main camera on the head with a resolution of 1280 x 800 pixels and two less camera, each in a end-effector, with 640 x 400 pixels. Each camera can record in 30 frames per second and can be used in real time by Baxter to perform tasks.

Baxter is connected to a computer using Ethernet, with a panic button in order to stop the robot in case of emergency. The software used on the communication between computer and Baxter is ROS, and allows to execute custom programs and those included by the robot itself (like to perform demonstrations, move to coordinates, etc).



The arm joints are named in the following manner:

S0 - Shoulder Roll
S1 - Shoulder Pitch
E0 - Elbow Roll
E1 - Elbow Pitch
W0 - Wrist Roll
W1 - Wrist Pitch
W2 - Wrist Roll

FIGURE 3.3: Baxters' joints.

Figure 3.3 shows the joints in a Baxters' arm[27]. Can be seen that there are 7 different joins: two on the shoulder (*S0* and *S1*), two on the elbow (*E0* and *E1*) and three on the wrist (*W0*, *W1* and *W0*). This joints are the ones that the Inverse Kinematics will compute, and in addition with time, their values are computed by the system to perform the required mean path.

3.3 ROS

The Robot Operating System[28] (ROS) is a set of software libraries and tools that help programmers to build robot applications for all kind of compatible robots. ROS provides standard operating system services such as hardware abstraction, low-level device control, implementation of commonly used functionality (like transformation and mathematical operations), message-passing between processes (between ROS nodes, used to execute programs) and package management. The main ROS client libraries are built in C++, Python and LISP, and are geared toward a Unix-like system because of their dependence on large collections of open-source software dependencies. Moreover, ROS allows to build software for more devices, like Polhemus Fastrak (used to track objects).

ROS encapsulates each program in a ROS node, allowing to communicate between them. With this system, in order to communicate with the Baxter, a node is created to perform the communications between the Baxter and the main program, and a second node encapsulates this main program, sending the instructions to the communication node. This architecture is useful to abstract the program from all the communications and its protocols, avoiding to program and deal with all that work.

In fact, the codes that used ROS are usual codes that calls ROS libraries and that are execute by the *rosrun* command on the command line. On the core of ROS runs Python, therefore only is necessary to compile the implemented code if this code is in C++, because Python and Lisp are interpreted languages.

3.4 Polhemus fastrak

Polhemus Fastrak[29, 30] is a motion tracker capable of delivering real-time, 6 degrees of freedom (DOF) tracking that is reliable, accurate and has an operational range of approximately 150 cm with the standard source. The tracker is connected to the computer through USB and all the communications are performed using ROS. In addition of the coordinates, the tracker is capable of sending the orientation of all the sensors, allowing to save and use all the information of the movements performed. The coordinates and rotations have the origin of the coordinate axis in the electromagnetic emitter of the tracker and therefore, in this thesis, the emitter will be located on the demonstrator.

Figure 3.4 shows one of the sensors used by the tracker, the emitter and the tracker itself. In addition, there is a pair of gloves to connect the sensors and track the hand,

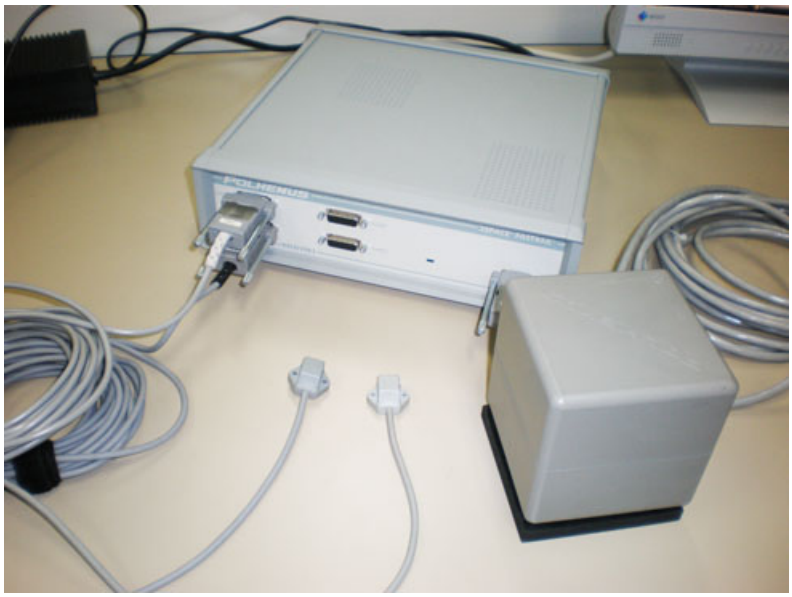


FIGURE 3.4: Sensors, emitter and Polhemus Fastrak itself.

but for this thesis they are not used. The latency is of 4 milliseconds, the static accuracy is 0'762 mm for the X, Y, or Z position and 0.15 degrees for receiver orientation. The tracker will provide the specified performance when the receivers are within 77 cm far from the transmitter.

3.5 Flask

Flask[31] is a microframework for Python that allows to build in simple way powerful web applications without particular tools or libraries. The main motivation for using Flask in this thesis is to build a simple web server capable to receive *http* petitions and to process them efficiently. Flask will encapsulate all the scripts used to compute the mean path, the Inverse Kinematics and the experiments for this thesis.

Chapter 4

Experimentation

As explained before, this thesis divides the main problem into two subproblems, trying to improve the knowledge learned applying reinforcement learning in both subproblems. Therefore, in order to solve them, the experimentation is divided in two parts. However, the same experiments will be used for testing both configurations (using end-effector position and using position with orientation). Hence, because the arm tracking is the same for both subproblems, the section that explains how to compute both position and orientation will obviate the demonstration capture.

It is important to point out that the first two sections of this experimentation focuses on how correctly choose the best algorithm of learning, how to ingest new demonstrations (in order to apply reinforcement learning) and how the user interacts with the system. In addition, the last section will give an overview of the environment where experiments are performed.

4.1 Computing the path from position

This section focuses on how to compute the mean path with the best algorithm for a new task using only the end-effector position and will show how to add new demonstrations in order to apply reinforcement learning. However, as explained before, in order to use the same demonstrations for both experimentation (position and position with orientation), both joints will be tracked. The first step is to attach both sensors in the hand and the elbow like in figure 4.1. It is important how a sensor is attached in order to correctly capture its position.

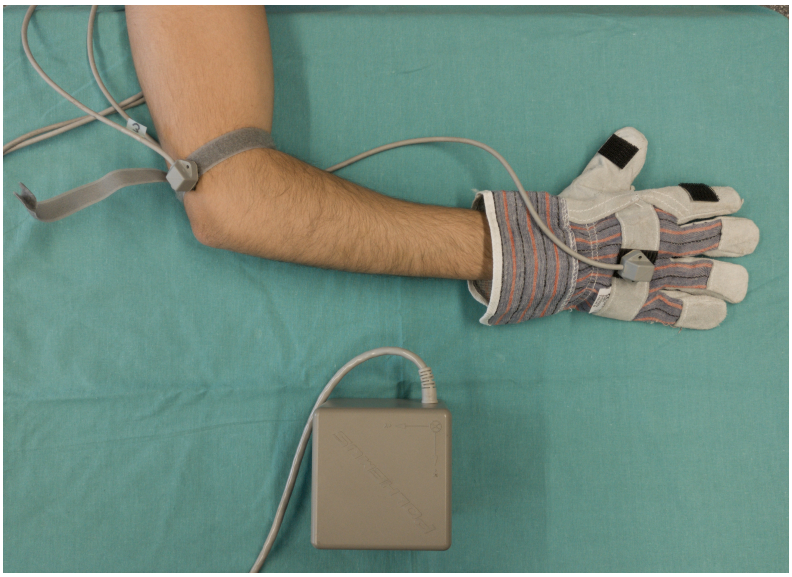


FIGURE 4.1: Emitter and sensors attached on the demonstrator arm.

The second step is to turn on the tracker and to execute the command (figure 4.2) to capture all the movements from both sensors. For testing purposes, figure 4.2 shows on the console the coordinates captured from the sensor, but all this information is saved on a file. Because the system is time-independent, the number of coordinates (i.e. the time tracking the demonstration) is not relevant whenever all (and only) the demonstration is captured. Therefore, it is important to start tracking a few seconds before (2 seconds is enough) starting the demonstration and stop tracking just when the demonstration is finished, avoiding all the unnecessary movement from the hand. This tracking will be performed **n** times, storing each demonstration on a different file. Therefore, when step two is completed, there will be a set of demonstrations **S** of size **n**. As can be seen in figure 4.2, the code is running in a ROS node, being necessary to start a ROS core in another command line.

```
kuka@kuka-pc:~/catkin_ws$ rosrun fastrak test_fastrak_publisher
Connected to fastrak over USB
1,0.76,34.30,-22.53
2,1.70,34.46,-21.41
1,0.76,34.30,-22.53
2,1.69,34.47,-21.40
1,0.76,34.31,-22.53
2,1.70,34.47,-21.41
1,0.76,34.31,-22.53
2,1.69,34.47,-21.40
1,0.76,34.31,-22.53
2,1.70,34.47,-21.40
1,0.76,34.31,-22.53
2,1.69,34.47,-21.41
1,0.76,34.31,-22.53
2,1.70,34.47,-21.41
```

FIGURE 4.2: Coordenates captured using Polhemus Fastrack.

On the third step, the demonstrations **S** are sent to the Flask server. This operation can be performed in two different ways: through a *curl* command or copying the demonstrations on a folder inside the server workspace (the server will detect the new demonstrations). Figure 4.3 shows the commands that the server will execute in order to compute the mean path with the different algorithms explained in Chapter 2, evaluating the accuracy of each path. Because we are dealing with the first demonstrations for for that new task, reinforcement learning can not be still applied (**S** are the base training set, therefore when new demonstrations are introduced, the system will apply reinforcement learning).

```
folder='pick-place'

# Geometric
folder = 'espiral-horizontal2'
all_points = trayectory.get_samples(folder)
sorted_all_points = all_points[all_points[:,3].argsort()]

mean_path_geo_mean_25, orientation_geo_mean_25 = trayectory.compute_medium_geometric(sorted_all_points, 25)
mean_path_geo_mean_50, orientation_geo_mean_50 = trayectory.compute_medium_geometric(sorted_all_points, 50)
mean_path_geo_mean_75, orientation_geo_mean_75 = trayectory.compute_medium_geometric(sorted_all_points, 75)
mean_path_geo_mean_100, orientation_geo_mean_100 = trayectory.compute_medium_geometric(sorted_all_points, 100)

compute_evaluation_geometric_mean(folder, mean_path_geo_mean_25, orientation_geo_mean_25,mean_path_geo_mean_50,
        orientation_geo_mean_50,mean_path_geo_mean_75, orientation_geo_mean_75,mean_path_geo_mean_100,
        orientation_geo_mean_100)


#GMM:
mean_path_gmm_25, orientation_gmm_25 = trayectory.GaussianMixture(25,folder)
mean_path_gmm_50, orientation_gmm_50 = trayectory.GaussianMixture(50,folder)
mean_path_gmm_75, orientation_gmm_75 = trayectory.GaussianMixture(75,folder)
mean_path_gmm_100, orientation_gmm_100 = trayectory.GaussianMixture(100,folder)

compute_evaluation_gmm(folder, mean_path_gmm_25, orientation_gmm_25, mean_path_gmm_50, orientation_gmm_50,
         mean_path_gmm_75, orientation_gmm_75, mean_path_gmm_100, orientation_gmm_100 )


#DTW online:
mean_path_dtw_online, orientation_dtw_online  = trayectory.compute_dtw_oneVSall(folder)

compute_evaluation_dtw(folder, mean_path_dtw_online, orientation_dtw_online)


#DTW hierarchical:
mean_path_dtw_hierarchical, orientation_dtw_hierarchical = trayectory.compute_dtw_hierarchical(folder)

compute_evaluation_dtw(folder, mean_path_dtw_hierarchical, orientation_dtw_hierarchical)
```

FIGURE 4.3: Commands executed by the server to compute mean paths and evaluate them.

Inside the server, when a new task is received, the system will compute all the mean paths, comparing their accuracies and choosing the best algorithm. The mean path from this selected algorithm will be send to a ROS node to compute the Inverse Kinematics[32–34]. The command executed internally by the server to send this information to the ROS node can be seen in figure 4.4. This command will return whether the path given is valid, in addition with the joints values or an error. It can be seen how the service sets a header, the position and orientation for the end-effector, the seed angles for the joints values (i.e. where the IK can start to compute the values) and the type of seed (starting the computation with the default values, the given ones or the actual position).

When the mean path is returned in the Baxters' format (i.e. joint values), the user can send this information to the robot starting a sending node to the Baxter and a node to

```
$ rosservice call /ExternalTools/left/PositionKinematicsNode/IKService "pose_stamp:
- header:
    seq: 0
    stamp: now
    frame_id: 'base'
  pose:
    position:
      x: 0.657579481614
      y: 0.851981417433
      z: 0.0388352386502
    orientation:
      x: -0.366894936773
      y: 0.885980397775
      z: 0.108155782462
      w: 0.262162481772
seed_angles:
- header: auto
  name: ['left_e0', 'left_e1', 'left_s0', 'left_s1', 'left_w0', 'left_w1', 'left_w2']
  position: [0.4371845240478516, 1.8419274289489747, 0.4981602602966309, -1.3483691110107423,
-0.11850001572875977, 1.18768462366333, -0.002300971179199219]
seed_mode: 0"
```

FIGURE 4.4: IK service.

read the file with the values[35] (both commands can be seen in figure 4.5).

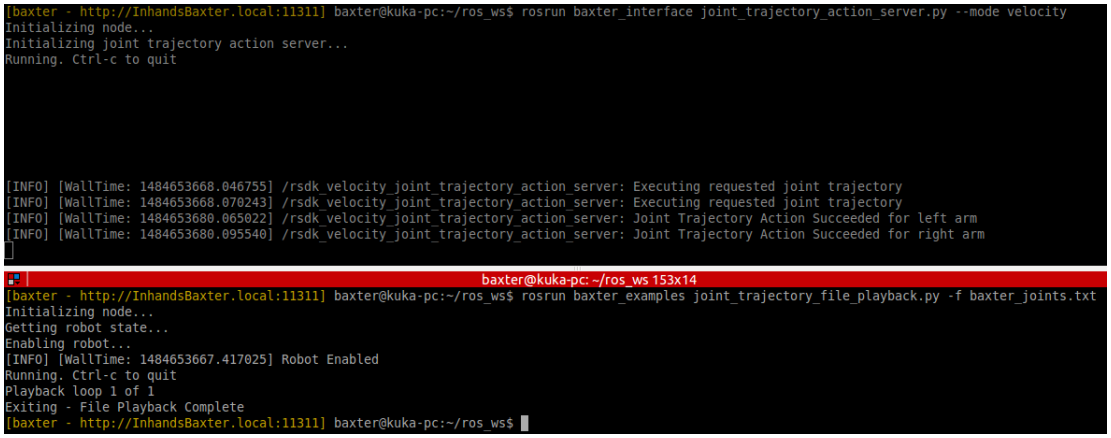


FIGURE 4.5: IK service.

At this point, if the robot can perform that task, the system will have a base training set and a mean path. Therefore, new demonstrations can be sent in order to learn from them through reinforcement learning. Hence, the fourth step consist in tracking a new demonstration of the task like in the first step and ingest it into the system. The system will return the mean path computed using the best algorithm (the one chosen in step 1) in values per joint and time for the Baxter. As explained in the *Reinforcemenent Learning* section, in Chapter 2, in order to learn from a new demonstration, it is required to get a qualification smaller than the chosen threshold. The threshold used in this project is a 25% of similarity between the new demonstration and the set of demonstrations that the server has for that task.

Therefore, following is an outlined overview of how to add a new task with a base training set and new demonstrations for that task:

1. Place both sensors on the human arm

2. Capture a set of demonstrations **S** from a new task
3. Send this set of demonstrations **S** to the Flask server
4. Send to the server new demonstrations of that task to be added through reinforcement learning

As explained, on step 3 and 4, the returned file can be directly sent to the Baxter using the ROS commands of Figure 4.5.

4.2 Computing the path from position and orientation

This section is pretty similar to the previous one, because on the previous experimentation, both hand and elbow are tracked, being computed the end-effector orientation using the second joint.

In addition to compute the mean path for the end-effector, the system computes the orientation for this end-effector using the coordinates tracked for both joints. Therefore, while computing the mean path between demonstrations, the system will compute the end-effector orientation for each demonstration, applying then the algorithms over path and orientation, returning the mean path and orientation (as can be seen in figure 4.3). For all mean paths, their evaluation will be computed, performing the same choice than the previous experimentation and using the algorithm with best accuracy. Furthermore, the IK will be computed using both mean path and orientation, returning the computed values for the robots arm joints.

4.3 Environment

In order to perform all the demonstrations in the same conditions, the same environment was set up. The experiments are performed on the environment shown in figure 4.6. Both demonstration and Baxters' execution are performed on the same position in order to maintain the same environment for both.

For simplicity, the experiments are performed with the right arm for demonstrator and robot, but they could be done with the left arm with no need to change anything. Figure 4.1 shows how the sensors are placed on the demonstrator arm. In addition, it can be seen the position of the emitter that will track the sensors.

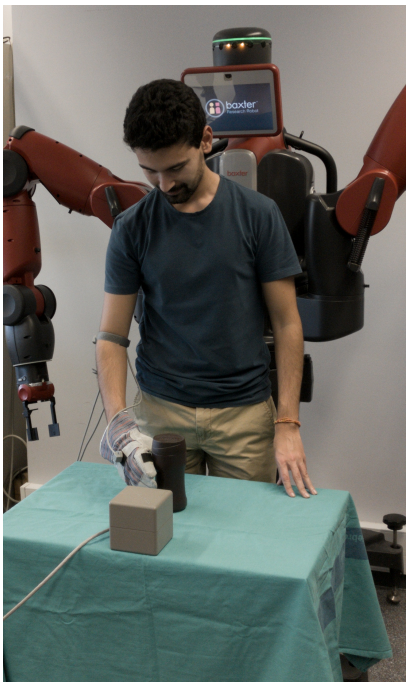


FIGURE 4.6: Demonstrator performing a demonstration.

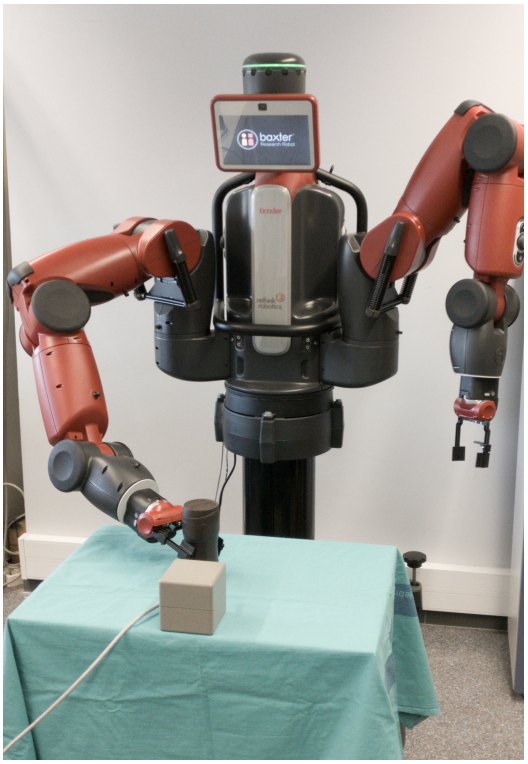


FIGURE 4.7: Baxter performing a task in a demonstration enviorement.

Figure 4.7 shows Baxter performing a task in the same environment than the demonstrator. It can be seen that the position is the same and that the emitter is placed in the same position. The emitter is the origin axis of coordinates, therefore is used to place demonstrator an robot in the same position. Because the emitter is not tn the robots' origin axis, a translation is needed before executing any computation.

Chapter 5

Results

The experimentation performed follows the methodology explained in Chapter 4, explaining step by step the obtained results. All algorithms are evaluated using the accuracy explained in Chapter 2, but only the mean path from the best algorithm will be executed by the robot. The reason behind this restriction is that, as will be explained below, the mean path resulting from the geometric mean and GMM are not safe enough. The first section will explain the experiment performed to compute the mean path using only the end-effector position, being section 2 the one that explains the experiments for computing the mean path from position and orientation.

5.1 Computing the path from position

As explained in Chapter 4, the first step is to track the demonstrators' arm tracking both hand and elbow. Figure 5.1 shows the demonstrator performing a set of demonstrations with both sensors attached.

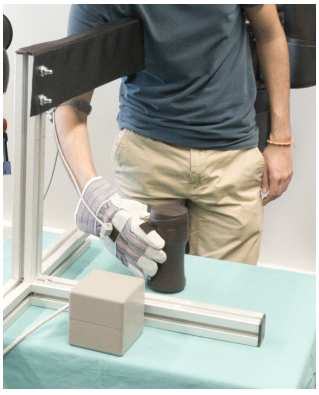


FIGURE 5.1: Demonstrator performing a demonstration.

The scenario is exactly the same for both demonstrator and robot. A set of 8 demonstrations are performed for a task of moving the arm under an obstacle. Moreover, it can be seen how the demonstrator is avoiding this object moving the whole arm under it.

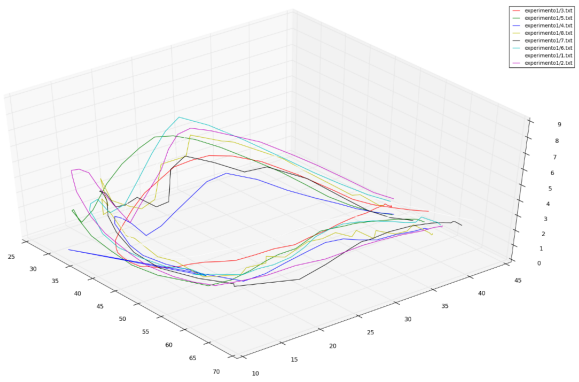


FIGURE 5.2: Trajectories tracked.

Figure 5.2 shows the different tracked trajectories for the end-effector. Clearly, all trajectories belong to the same task, being pretty similar between them. The start and the end points are not the same, though they are quite close. The movement is an ellipse where one of the ends of the major axis has a movement similar to an *S*. In front of this extreme, it can be seen how the Z axis decreases from 5 cm to 1 cm. It is important to point out that, for precision reasons, the blue trajectory (referred now as blue demonstration) has an outlier, causing that the path has an incorrect peak. At first instance it may seem that this demonstration contains errors, inot being recommendable its use, because it will worsen the resulting mean path. However, it will be used to see how the algorithms are able to respond to this errors. As explained in Chapter 2, the evaluation for the algorithms consist in comparing the distance of the computed mean path with the mean distance that the demonstrations have among them. Therefore, the computed mean distance among the demonstrations from figure 5.2 is of 122. This distance for this task means that, as can be seen in figure 5.2, the demonstration are pretty similar.

From these 8 demonstrations, a mean path per algorithm is computed. Figure 5.3 shows the resulting mean path from applying geometric mean with different number of bins. The white points are the ones that belong to the 8 demonstrations, being the blue line the mean path. The first mean path (the upper left image) is computed using 25 bins. It can be seen how the trajectory goes through the points of the other trajectories in straight lines, being clearly underfitting the trajectories (this path is generalist, passing through the middle but not getting adjusted). In fact, it seems that with 25 bins the algorithm does not even take into account the error that contains one of the demonstrations. The

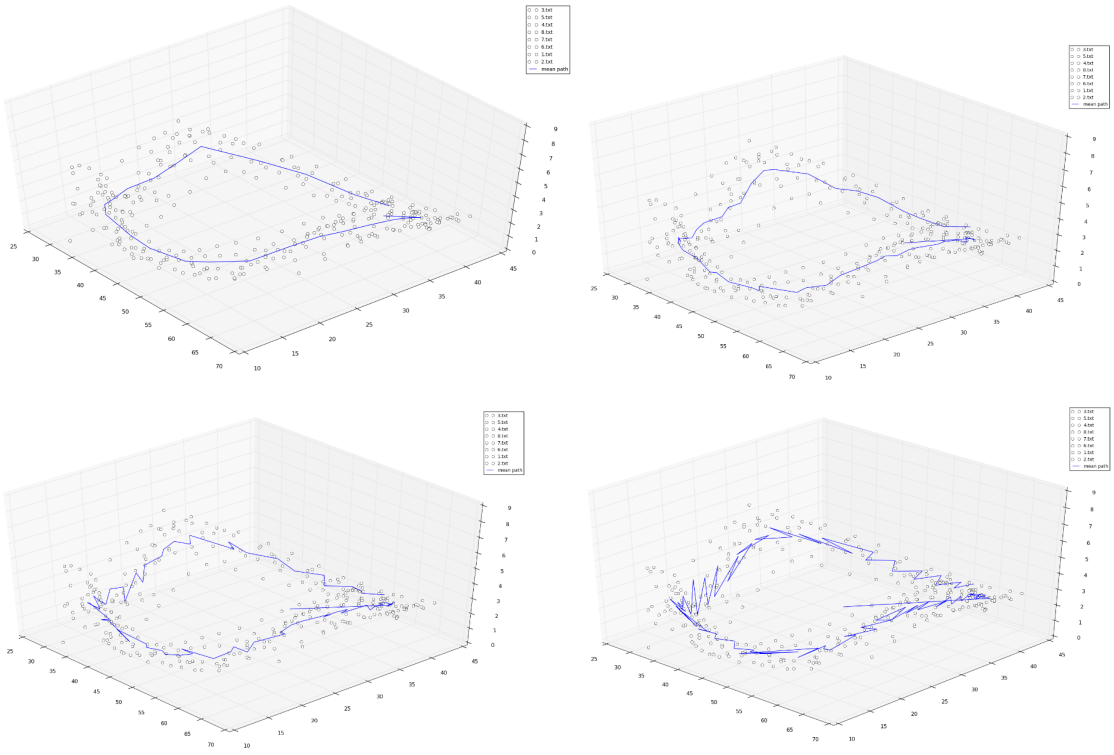


FIGURE 5.3: Mean paths computed with different size per bin using geometric mean. *Upper left figure shows the mean path with 25 bins, upper right figure shows the mean path with 50 bins, bottom left figure shows the mean path with 75 bins and bottom right figure shows the mean path with 100 bins.*

next mean path (the upper right image) is computed with 50 bins. In this case, it seems that the path starts to adjust better to the demonstrations, showing how the Z axis decreases before the S movement but still being too generalist. The next one (the bottom left image) is computed using 75 bins, being clearly overfitting with the trajectories. The last image shows the mean path computed with 100 bins, being a kind of zigzag that starts overfitting the trajectories. Clearly, this path is not safe to being executed by the robot, because the zigzag movement will result in a weird movement.

	25 bins	50 bins	75 bins	100 bins
DTW distance	88.76	101.71	136.4	179.7

TABLE 5.1: Accuracy per bin.

Table 5.1 shows the distance obtained per bin. As can be seen, all the distances computed are far from the mean distance of 122, where the mean path computed with 75 bins is the nearest mean path. It is curious to see that this mean path gets a distance more similar than the one computed with 50 bins, because with 75 bins the mean path presents a continuous movement in zigzag, while with 50 bins it seems more natural. The distances for the mean paths using 25 and 75 bins are pretty far from the others, because the first is underfitting and the last is overfitting. As explained before, the bins are computed using

spatial coordinates and time, therefore the mean path computed has spatial coordinates and time. This is the reason behind this type of trajectory, because the geometric mean computed includes an incorrect time per bin. The reason behind this *error* is that the geometric mean is not time independent, grouping per bin points from different time and creating this effect of zigzag move.

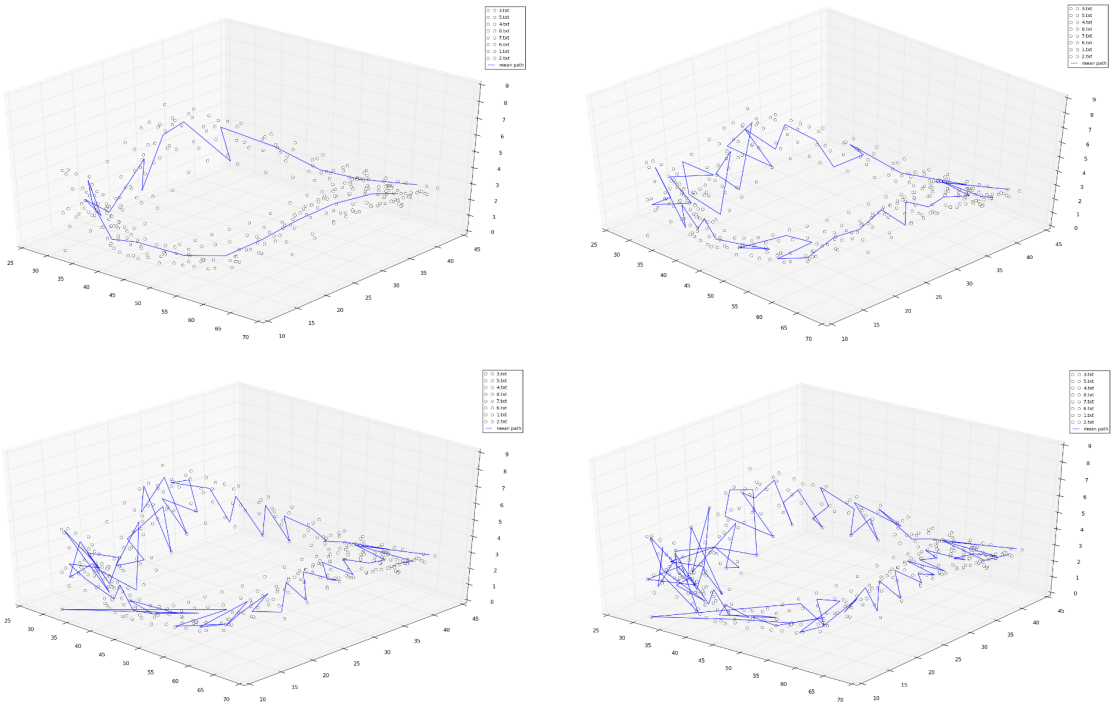


FIGURE 5.4: Mean paths computed with different size per bin using GMM. *Upper left figure shows the mean path with 25 bins, upper right figure shows the mean path with 50 bins, bottom left figure shows the mean path with 75 bins and bottom right figure shows the mean path with 100 bins.*

Figure 5.4 shows the mean path using GMM with the same bins used on the geometric mean. As can be seen, for the mean path computed with 25 bins (the upper left image) there are some sections with a movement in zigzag, similar to that of the previous algorithm but wider. In the second mean path (the upper right image), computed with 50 bins, it can be seen how it follows a zigzag movement even wider than with 25 bins. For 75 and 100 bins the zigzag movement is simply getting even bigger, because with more bins the algorithm is overfitting the demonstrations. In the case of the mean path computed with 100 bins, the path is overfitting so much that it catches exactly the same error as the blue trajectory in figure 5.2. It is important to point out that, as explained in Chapter 2, the GMM is fed with both spatial coordinates and time, not online spatial coordinates as commonly used. The addition of time leads the algorithm to try to fit the computed bins into space and time, creating this type of trajectory in zigzag and having the same problem than the geometric mean.

	25 bins	50 bins	75 bins	100 bins
DTW distance	90.85	160.26	204.52	290.39

TABLE 5.2: Accuracy per bin.

Table 5.2 shows the accuracy obtained per bin. As can be seen, the mean path with a distance more similar to the mean distance between demonstrations is the one computed with 25 bins. However, as commented before, the path is trying to overfit to the demonstrations, following the discussed zigzag.

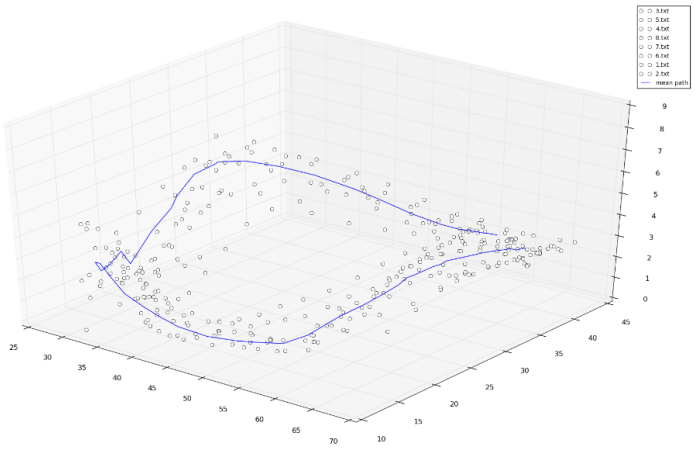


FIGURE 5.5: Mean path computed using DTW with online joining.

The resulting mean path using DTW with an online joining can be seen in Figure 5.5. Unlike the mean paths computed with the geometric mean and GMM, DTW seems to find a good mean path adjusted to the trajectories without overfitting. It can be seen that the error contained in the blue demonstration in figure 5.2 is not affecting the mean path computed. This happens because in the online joining some demonstrations have less importance than others, disappearing, in this case, that error. However, if the blue trajectory would have been computed the last, the final mean path would contain a large pick in the same position than the blue demonstration. It can be seen how this algorithm is correctly computing the mean path containing the main features from all trajectories. For example, the mentioned movement describing a S is also present in this mean path, with some fluctuations and a previous decreasing on the Z axis. The overall mean path, in addition to how well seems to be adjusted to the trajectories, presents a movement pretty similar to a human movement, i.e. without strange fluctuations and being fluid. However, as commented before, the order in which demonstrations are processed will affect the final mean path, leading maybe to a bad solution. The distance that gets this mean path with the demonstrations is 115.5, coming near the 122 from the demonstrations among them.

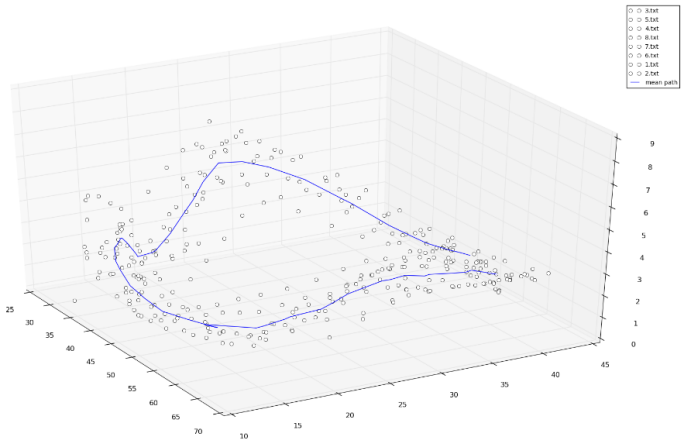


FIGURE 5.6: Mean path computed using DTW with hierarchical joining.

Figure 5.6 shows the resulting path from using DTW with a hierarchical joining. As can be seen, this mean path presents a small error in the same section than error of the blue demonstration is. This precision error shows how the DTW with hierarchical joining can reduce errors that are not repeated in all the demonstrations. Furthermore, it can be seen how the mean path is pretty similar to all the demonstrations, following the same type of movement and the main characteristics. Unlike geometric mean and GMM, DTW is not trying to overfit the demonstrations with a zigzag movement, meaning that the coordinates computed are in consonance with the instant of time they belong to. The distance that gets this mean path with the demonstrations is 119.7, being the algorithm that reaches a more similar distance that the demonstrations have among them.

	Geoemtric Mean	GMM	DTW online	DTW hierarchical
Accuracy	136.4 (75 bins)	90.85 (25 bins)	115.5	119.7

TABLE 5.3: Accuracy per each algorithm.

Table 5.3 shows the resulting accuracy per algorithm. In the case of geometric mean and GMM, the accuracy shown in the table is the best one obtained.

Figure 5.7 shows the mean paths with best accuracy per algorithm. As can be seen, the results computed by DTW (both joinings) significantly improve the results obtained with geometric mean and GMM. While these two methods can not compute an acceptable trajectory for the end-effector, DTW computes a path adjusted to the demonstrations, keeping the key features of the trajectories but without overfit nor perform a zigzag movement. Furthermore, as can be seen in figure 5.7, the computed mean paths are safe to perform by the robot (because the trajectories do not have weird movements).

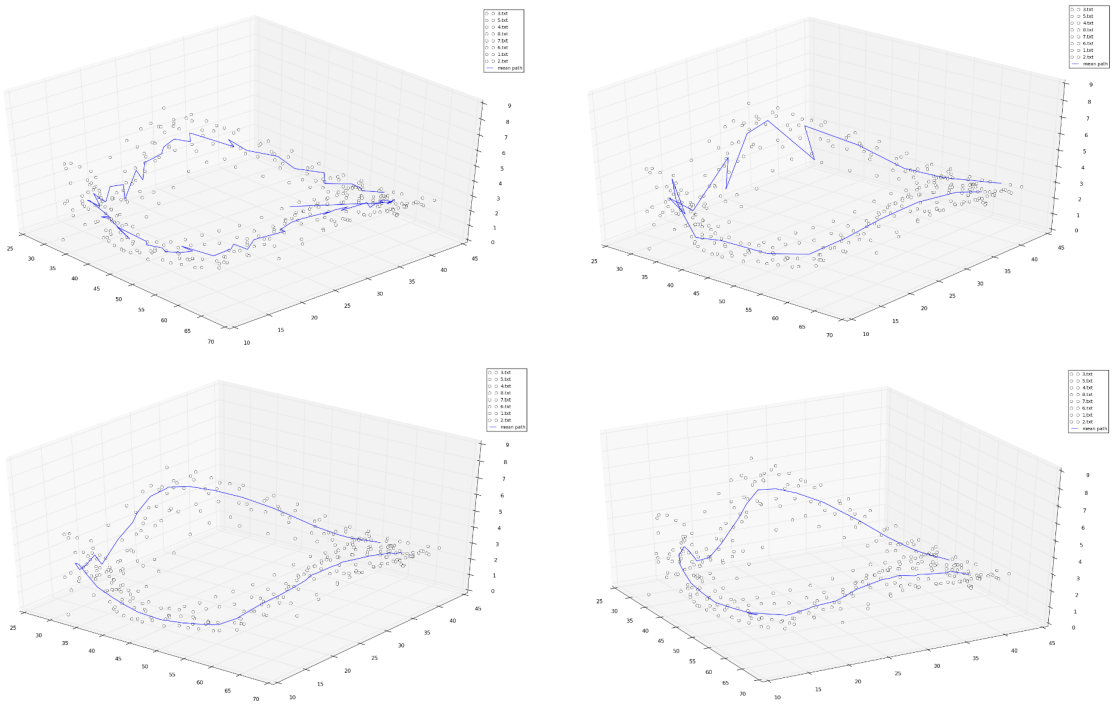


FIGURE 5.7: Mean paths computed per algorithm.

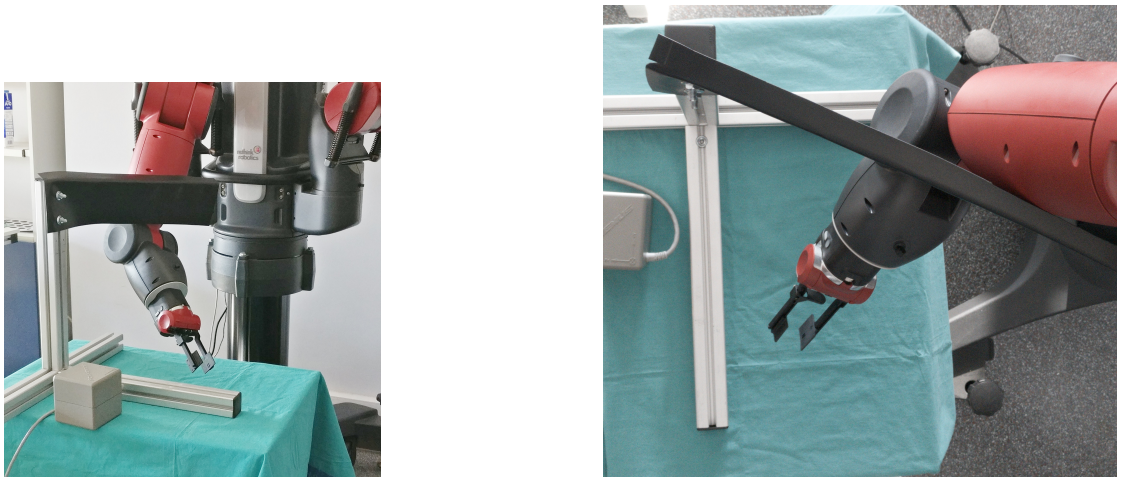


FIGURE 5.8: Baxter executing the mean path computed using DTW with hierarchical joining.

Figure 5.8 shows the robot performing the mean path computed using DTW with hierarchical joining. As can be seen, the end effector is under the obstacle, mimicking the movement performed by the hand of the demonstrator, but because the end-effector orientation, the whole arm is hitting the obstacle. This example shows perfectly that, in spite of solving the *imitation* subproblem, the correspondence between human and robot is not correct, hitting the obstacle to be avoided.

The demonstration performed uses as obstacle a flexible plastic, but if the obstacle were a human arm, the robot would hit and push the human arm without stopping. Moreover,

if the obstacle were a fragile object or element in an assembly line, probably the object would break or damage.

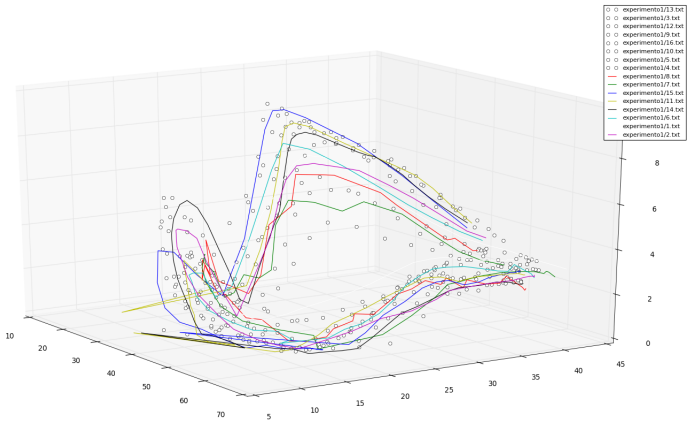


FIGURE 5.9: All demonstrations performed. *In white points the previous demonstrations and in coloured trajectories the new ones.*

In order to show how reinforcement learning can improve the current mean path, a set of 8 more demonstration has been captured. However, this 8 demonstration are slightly different from the previous 8. As can be seen in figure 5.9, the new demonstrations (the coloured trajectories) are a little above the other demonstrations. This difference is intended, allowing to see how the mean path will tend to these new set of demonstrations. Moreover, it can be seen how some of these new demonstrations have some precision errors, having peacks in the middle of their trajectories. Therefore, the mean path using DTW with hierarchical and online joining are shown in figure 5.10 and figure 5.11 respectively.

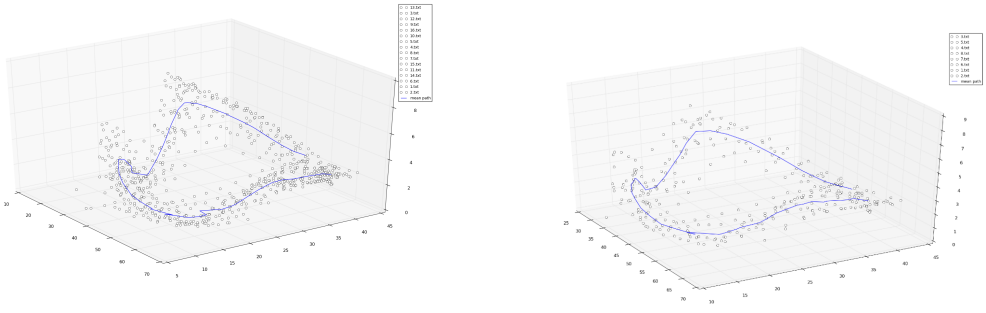


FIGURE 5.10: Mean path using hierarchical joining with reinforcement demonstrations on the left and the 8 first demonstrations on the right

As can be seen on figure 5.10, the new mean path computed (on the left) with DTW and hierarchical joining is still adjusted to the trajectories, but it is slightly different than the previous mean path (on the right). This difference is mainly because the new demonstrations are similar among them but sightly different than the first 8 demonstrations. Therefore, new demonstrations can be used to apply smalls differences in a task in order

to modify the computed mean path. This differences may appear if the demonstrator improves his own performance for that task, and therefore, want to improve the task performed by the robot.

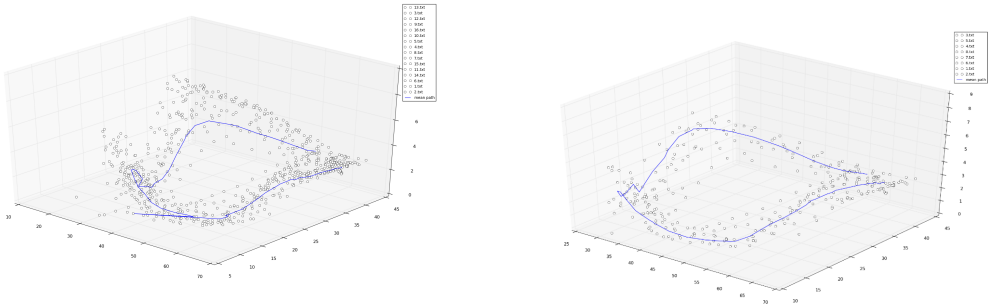


FIGURE 5.11: Mean path using online joining with reinforcement demonstrations on the left and the 8 first demonstrations on the right

Figure 5.11 shows the computed new mean path using DTW with online joining. As can be seen, the mean path computed (on the left) has changed compared to the previous one (on the right). In fact, the error detected in one of the previous demonstrations (the blue demonstration) appears now in this mean path. This happens because, as explained en Chapter 2 and discussed before, in the online joining some demonstrations have more importance than others in the computation of the mean path. In this case, the first demonstrations are joined later than the new demonstrations, biasing the mean path to the previous demonstrations.

5.2 Computing the path from position and orientation

Computing the orientation of the end-effector is done per algorithm along with the mean path. As explained in Chapter 2, the evaluation is applied for the mean path, comparing how different it is to the demonstrations. However, the evaluation of how the orientation affects to the robots' arm has to be performed visually.

Figure 5.12 shows a single demonstration with the orientation in different points. As can be seen, at the beginning of the task, the end-effector is not pointing at the direction of the movement, orienting the end-effector while the demonstration advances. Because the elbow is tracked in the same way for all demonstrations, the orientation between them is pretty similar.

The experimentation using orientation in addition with position is pretty similar on methodology, being the mean path computed for the end-effector the same. The 8

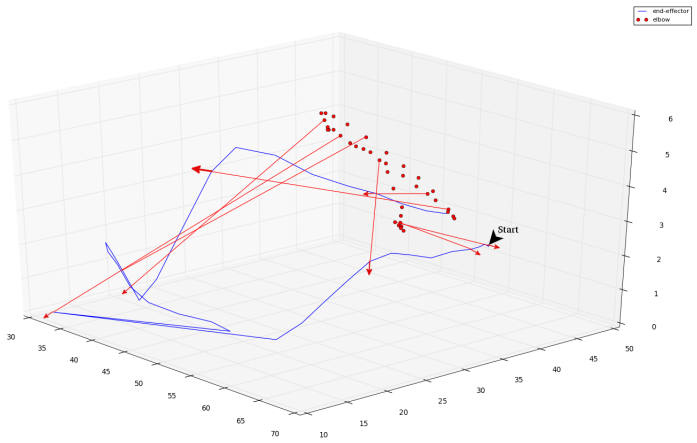


FIGURE 5.12: A demonstration with orientation. *End-effector trajectory in blue, elbow trajectory in red points and orientation in red arrows.*

demonstrations used in the experiments for position will be used for position with orientation. Therefore, the graphics are pretty similar, with the addition of the orientation in different points.

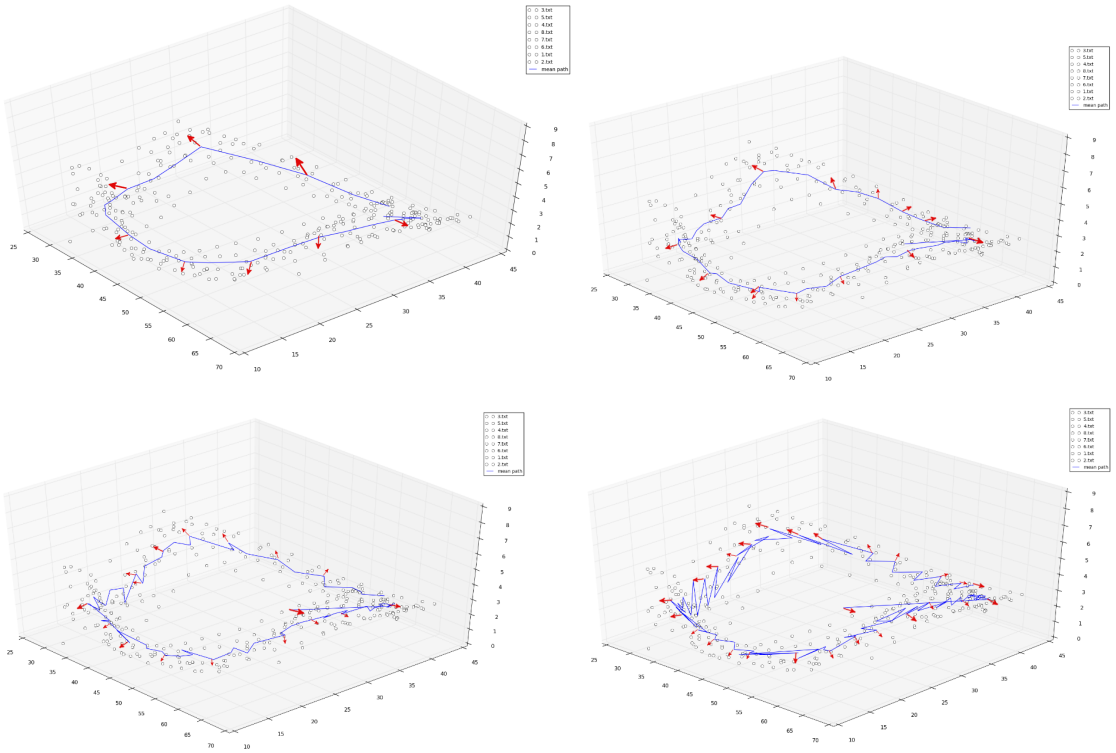


FIGURE 5.13: Mean paths computed with different size per bin using geometric mean with position and orientation. *Upper left figure shows the mean path with 25 bins, upper right figure shows the mean path with 50 bins, bottom left figure shows the mean path with 75 bins and bottom right figure shows the mean path with 25 bins.*

Figure 5.13 shows the orientation computed for the different mean paths with geometric mean. As can be seen, the orientation in each bin is similar to the demonstrations, because the orientation among demonstrations is pretty similar. Therefore, the orientation

computed is good enough, being able to be used along the mean path if it were not so fluctuating. In this case, because the distance between hand and elbow along with the similar orientation among demonstrations, the grouping in bins using spatial coordinates with time is not creating fluctuations on the orientation. However, if the orientations among demonstrations were not similar (i.e. varying among demonstrations), the final orientation computed with this algorithm would vary enough to stop being safe to be executed by the robot.

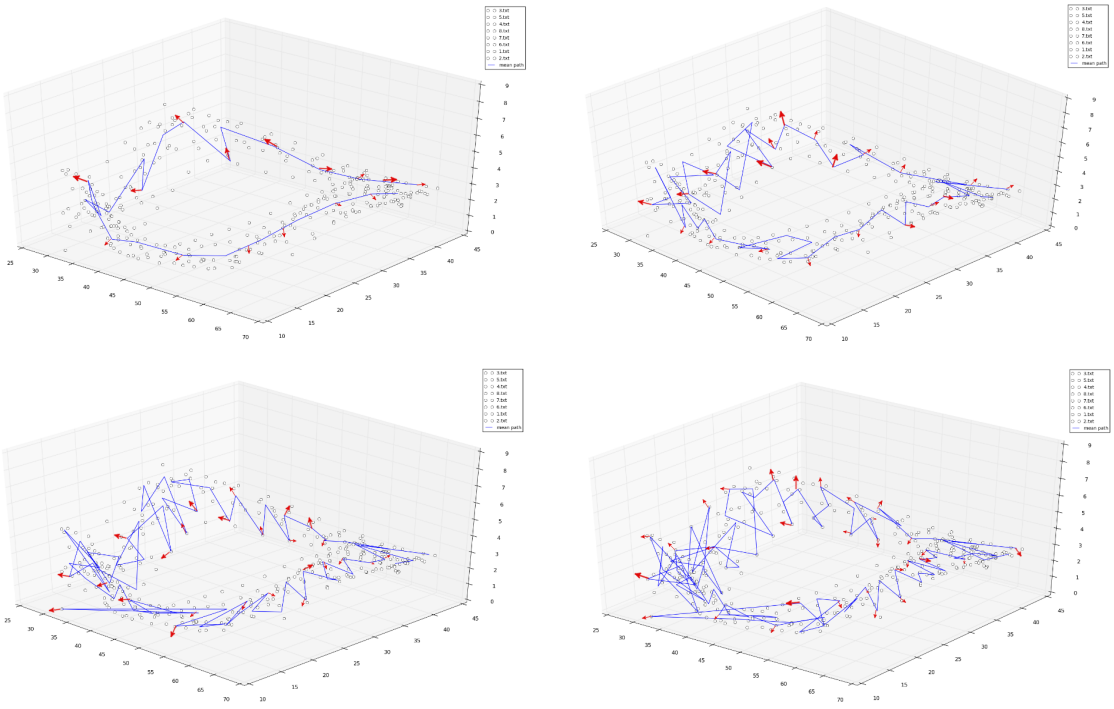


FIGURE 5.14: Mean paths computed with different size per bin using GMM with position and orientation. *Upper left figure shows the mean path with 25 bins, upper right figure shows the mean path with 50 bins, bottom left figure shows the mean path with 75 bins and bottom right figure shows the mean path with 100 bins.*

Figure 5.14 shows the mean path computed using GMM with the orientation. As can be seen for all graphics, the orientation is fluctuating along points, without keeping the correct orientation nor a similar one. In fact, the orientation seems to fluctuate as much as the mean path, being not safe to execute them on the robot. The reason behind this fluctuations on the orientation is the same that for the mean path fluctuation: the orientation is computed using spatial coordinates along with time. Only the orientation for the left section of the trajectory seems to keep a similar orientation, having the others sections too many fluctuations.

The resulting mean path using DTW with an online joining can be seen in Figure 5.15. As one might suppose, DTW is keeping a similar orientation than the demonstrations in each coordinate, having practically the same orientation than them. As discussed in

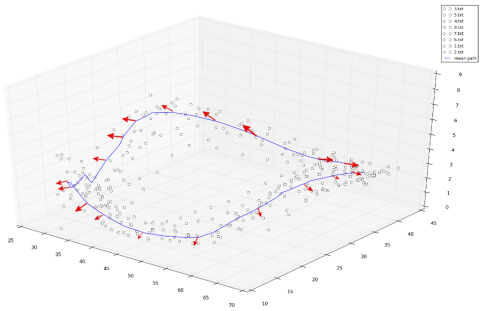


FIGURE 5.15: Mean path computed using DTW with online joining with position and orientation.

Chapter 2, DTW computes the mean point between two points, computing the orientation in the same way. Because the orientation is similar among demonstrations, the computed orientation on the mean path will remain similar. Furthermore, because the demonstration with the error is one of the first to be joined, the final orientation in those points is not affected.

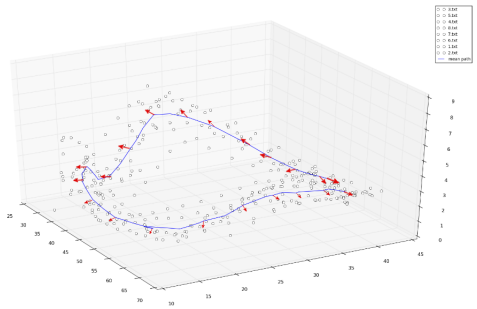


FIGURE 5.16: Mean path computed using DTW with hierarchical joining with position and orientation.

Figure 5.16 shows the resulting path from DTW with a hierarchical joining. In the same way than the online joining, with hierarchical joining the orientation is pretty similar to the demonstrations. Furthermore, it can be seen how the orientations on the different points are more similar to figure 5.12 than the ones computed by the online joining. Therefore, in addition to compute the best mean path, it is computing a more similar orientation for the end-effector than the online joining.

Figure 5.17 shows the mean paths with best accuracy per algorithm using position and orientation. As disscused before, the best orientation is performed by the DTW with hierarchical joining, followed by the online joining (pretty similar) and the geometric mean, being the three of them goods orientations to use with the robot. However, GMM is not computing a good orientation, having this too many fluctuations to be used by the robot.

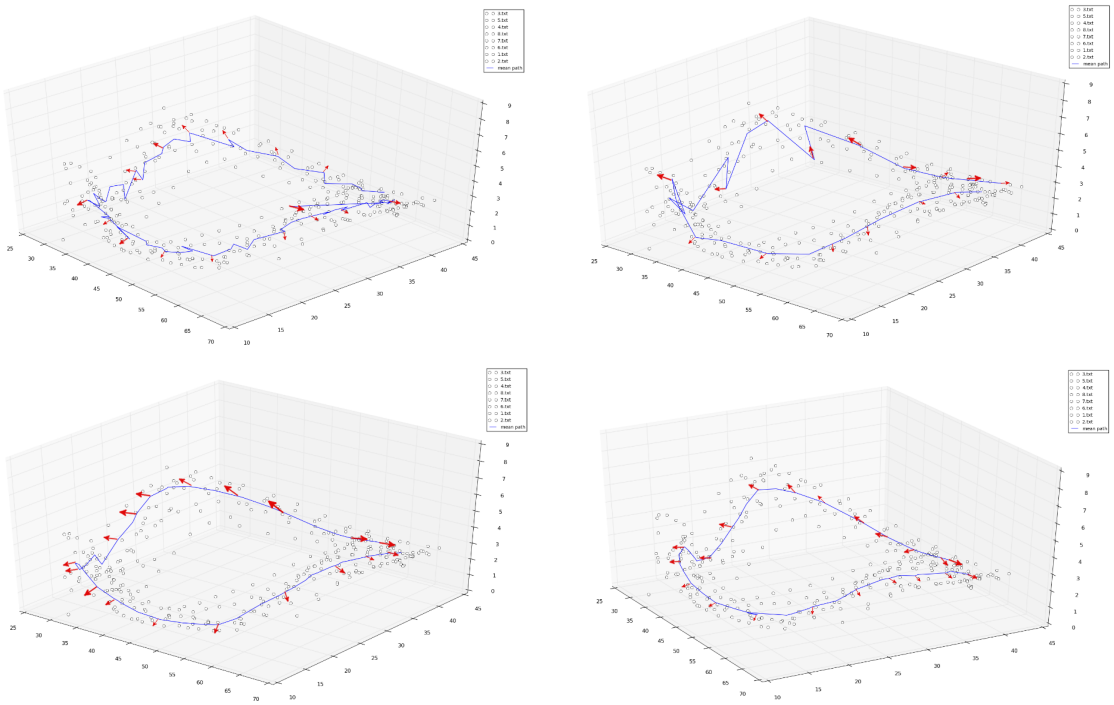


FIGURE 5.17: Mean paths computed per algorithm with position and orientation. *Upper left figure shows the mean path using geometric mean, upper right figure shows the mean path using GMM, bottom left figure shows the mean path using DTW with online joining and bottom right figure shows the mean path using DTW with hierarchical joining.*

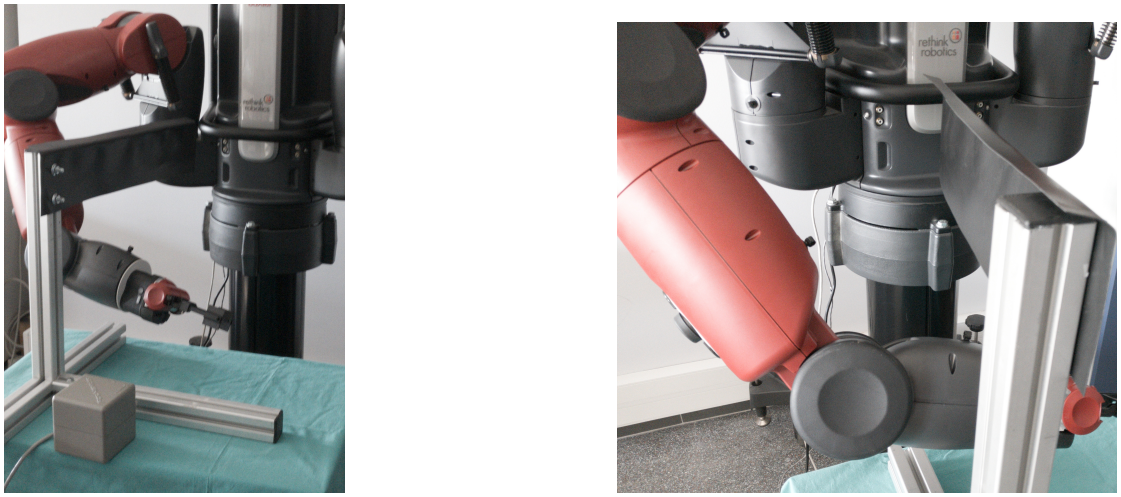


FIGURE 5.18: Baxter executing the mean path computed using DTW with hierarchical joining with position and orientation.

Figure 5.18 shows the robot performing the mean path computed using DTW with hierarchical joining with position and orientation. Both pictures are showing from different perspectives how in this case the robot is avoiding the obstacle moving the forearm behind it. As can be seen in figure 5.16, the orientation for the end-effector matches with the orientations in this mean path.

Chapter 6

Conclusions

The experimentation performed has show different conclusions per algorithm, problem and subproblems. The main problem faced in this thesis was **how to facilitate the learning of new tasks by robots through demonstrations performed by humans**, facing two different subproblems: *imitation* and *correspondence*.

The first one was solved easily using DTW with both joining techniques: both techniques reached a good accuracy for the mean path computed. However, both geometric mean and GMM turned out to have problems to compute the mean path. As shown in the experiments, all the mean path computed with the different bins had fluctuations that impeded that the result was acceptable. The main problems behind both algorithms are that they depend on the number of bins selected and, given the way that they compute the paths, the resulting mean path is not approaching an acceptable trajectory. As explained in Chapter 2, both algorithms will group the points of the trajectories in bins, losing all the information of the trajectories followed in the demonstrations. This information is lost because it is possible to join points that are near in time but distant in space, resulting in a point that in fact is the mean of those points, but it is not similar to any trajectory followed. On the contrary, DTW computes a mean path that correctly approximates to the demonstrations performed. With the mean path computed using DTW with hierarchical joining, the robot was able to correctly perform the task performed by the human demonstrator. Finally, the hierarchical joining proved to reach more accuracy than the online joining. This improvement is because the importance of the demonstrations in the hierarchical joining is the same for all demonstrations, while with the online joining the last demonstrations will have more importance on the resulting mean path than the first demonstration processed.

For the second subproblem, the use of the elbow in order to compute the correct orientation of the end-effector turned out to be a success, with the arm performing in an anthropomorphic way the task. It is important to point out that the mean path computed is the same that for the previous subproblem, but this orientation improves all the execution, making it easier for the robot to work in human environments and performing tasks in a more anthropomorphic way. However, because all the joints do not have restrictions in it's positions, the robotic arm does not imitate perfectly the demonstrator arm. Therefore, in a more complex task, the computed values for the joints can lead the arm to reach positions that will hit some obstacle. Moreover, with more complex tasks, a better human-robot mapping "taking into account the whole demonstrators' arm" will be needed.

Another important fact to point out was the working space of the robot. This space will affect how the robot performs the computed mean paths, being impossible to perform some of these mean paths because the arm can not reach that position with the given orientation. Furthermore, the more the movement has to resemble an anthropomorphic movement, the more the difficult of mean path computing will be. This difficulty is due the restrictions added to the IK computation, being it more difficult to find the correct values for all the robots' joints. In spite of being a problem with the robot's structure and not with the system itself, further work can lead to improve the working space on the demonstrations, allowing the robot to reach positions that can not be reached by now.

Using Polhemus Fastrack has turned out to be a precise method to track joints on the demonstrator's arm. However, the wires connected to the sensor can block some of the movements performed by the arm, worsing the demonstration or forcing to repeat its repetition. Therefore, the improvement of this method is definitely a way to improve the whole system.

6.1 Future work

As shown in the experiments, with the current system, in addition to compute the mean path followed by the end-effector in a set of demonstrations, the values for the robots' joints can be computed with restrictions in order to move the robots' arm in an anthropomorphic way. However, different key steps of this thesis can be improved in order to compute better movements.

As stated before, the wires used by the sensors of the Polhemus Fastrack can block some demonstrations, being this field a key field to improve. A solution can be to use a remote sensor, like a Kinect camera, in order to track the demonstrators' arm joints, eliminating the need of using sensors to track demonstrations and facilitating how the demonstrator performs the movements. In addition, Kinect can be used to apply object recognition, helping the robot to better recognise the environment and to avoid possible obstacles in the trajectory.

The human-robot mapping is definitely another future work in order to improve how the robots perform a task. Only 2 joints are tracked correctly, using the second joint to compute the end-effector orientation. With more joints, a better mapping between robots' arm and demonstrators' arm can be performed, improving the movement of the whole arm and allowing to perform more complex tasks imitating the movement of the demonstrators' arm.

Related to this arm, a great improvement can be done with a custom implementation for the Inverse Kinematics. The IK used is the default for the robot, which, although good, could be improved by adding the option of setting the coordinates of each joint. With this improvement in union with an improvement on the human-robot mapping and tracking more joints on the demonstrators' arm, the robots' arm could perform the trajectory with a movement more similar to the demonstrators' arm.

A last improvement is related with the task itself: only one arm is considered to perform a task. It is very interesting to allow a second arm to perform a task, collaborating between them in the same way a demonstrator can do. This improvement allows to perform more advanced tasks that require the use of two or more arms (adding more robots).

Bibliography

[1] Ekvall, S.; Danica, Kragic D. *Robot Learning from Demonstration: A Task-Level Planning Approach*. 2004. [Online] <https://pdfs.semanticscholar.org/9a80/97e47fec9d37113ac48399719242c765a91f.pdf> [01/2017]

[2] Ekvall, S.; Danica, Kragic D. *Learning Task Models from Multiple Human Demonstrations*. 2004. [Online] <http://www.csc.kth.se/~danik/Papers/ekvallROMAN06.pdf> [01/2017]

[3] Maja J. *Getting Humanoids to Move and Imitate*. 2000. [Online] <https://robotics.usc.edu/publications/media/uploads/pubs/113.pdf> [01/2017]

[4] Ogawara, K.; Takamatsu, J.; Kimura, H.; Ikeuchi, K. *Generating a Symbolic Task Model from Multiple Demonstrations*. 2002. [Online] <http://www.cvl.iis.u-tokyo.ac.jp/CVL_research_2003/robo/ogawara.pdf> [01/2017]

[5] Chen, J.; Zelinsky, A. *Programing by Demonstration: Coping with Suboptimal Teaching Actions*. 2003. [Online] <http://users.cecs.anu.edu.au/~chen/publications/IJRR_2003.pdf> [01/2017]

[6] Schaal, S. *Learning From Demonstration*. [Online] <https://papers.nips.cc/paper/1224-learning-from-demonstration.pdf> [01/2017]

[7] Akgun, B.; Subramanian, K. *Robot Learning from Demonstration: Kinesthetic Teaching vs. Teleoperation*. 2011. [Online] <http://www.cc.gatech.edu/~ksubrama/files/HRIFinalBK.pdf> [01/2017]

[8] Akgun, B.; Cakmak, M.; Wook Yoo, J.; Thomaz, A. *Trajectories and Keyframes for Kinesthetic Teaching: A Human-Robot Interaction Perspective*. 2012. [Online] <http://www.cs.utexas.edu/users/sniekum/classes/RLFD-F16/papers/Akgun12.pdf> [01/2017]

[9] Kober, J.; Bagnell, A.; Peters, J. *Reinforcement Learning in Robotics: A Survey*. 2013. [Online] <http://www.ias.tu-darmstadt.de/uploads/Publications/Kober_IJRR_2013.pdf> [01/2017]

[10] Vibhute, V.; Angal, Y.; SURYAKANTH, B. *A survey of learning from demonstration used in robotic*. 2015. [Online] <https://www.ijsr.in/upload/1601896809NCRIET-246.pdf> [01/2017]

[11] Breazeal, C.; Scassellati, B. *Challenges in Building Robots That Imitate People*. 2000. [Online] <http://groups.csail.mit.edu/lbr/hrg/2000/MITPress-Imitation.pdf> [01/2017]

[12] Lopes, M.; Melo, F.; Montesano, L.; Santos-Victor, J. *Abstraction Levels for Robotic Imitation: Overview and Computational Approaches*. year. [Online] <https://flowers.inria.fr/mlopes/myrefs/10-SpringerBook-Imitation.pdf> [01/2017]

[13] name Mohammad, Y.; Nishida, T. *Tackling the Correspondence Problem Closed-Form Solution for Gesture Imitation by a Humanoid's Upper Body*. [Online] <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.418.8611&rep=rep1&type=pdf> [01/2017]

[14] Upcroft, B.; Kumar, S.; Ridley, N.; Ling, L;. Durrant-Whyte, H. *Fast Re-parameterisation of Gaussian Mixture Models for Robotics Applications*. [Online] <https://pdfs.semanticscholar.org/2280/a91442f85d5bc6f09fad0826e8365b01c25f.pdf> [01/2017]

[15] Chernova, S.; Veloso, M. *Confidence-Based Policy Learning from Demonstration Using Gaussian Mixture Models*. 2007. [Online] <http://www.cs.cmu.edu/~mmv/papers/07aamas-sonia.pdf> [01/2017]

[16] Chatterjee, A.; Rakshit, A.; Singh, N. *Visio based Autonomous robot navigation*. 2013

[17] Johnen, B.; Kuhlenkoetter, B. *A Dynamic Time Warping algorithm for industrial robot motion analysis*. 2016.

[18] Zhou, F.; De la Torre, F. *Generalized Time Warping for Multi-modal Alignment of Human Motion*. 2011. [Online] <http://www.ri.cmu.edu/pub_files/2012/6/gtw.pdf> [01/2017]

[19] Kopanicakova, A.; Vircikova, M.; Sincak, P. *Gesture Recognition using DTW and its Application Potential in Human-Centered Robotics*. 2014. [Online]

<http://neuron.tuke.sk/maria.vircik/phd/files/documents/publications/Gesture%20Recognition%20using%20DTW%20and%20its%20Application%20Potential%20in%20Human-Centered%20Robotics.pdf> [01/2017]

[20] Ji-Hyeong, H.; Jong-Hwan, K. *Consideration about the Application of Dynamic Time Warping to Human Hands Behavior Recognition for Human-Robot Interaction.* 2014.

[21] Reynolds, D. *Gaussian Mixture Models.* 2001. [Online] <https://pdfs.semanticscholar.org/734b/07b53c23f74a3b004d7fe341ae4fce462fc6.pdf> [01/2017]

[22] Liarokapis, M.; Charalampos P. Panagiotis, B.; Kyriakopoulos, J.; Artemiadis, K. *Directions, Methods and Metrics for Mapping Human to Robot Motion with Functional Anthropomorphism: A Review.* 2014. [Online] <http://www.minasliarokapis.com/TR2013_Liarokapis_MappingSkillTransfer.pdf> [01/2017]

[23] De Luca, A. *Inverse kinematics.* [Online] <http://www.diag.uniroma1.it/~deluca/rob1_en/10_InverseKinematics.pdf> [01/2017]

[24] Yao, M. *Mathematics for Inverse Kinematics.* [Online] <http://www.cs.cmu.edu/~15464-s13/lectures/lecture6/IK.pdf> [01/2017]

[25] *Baxter.* [Online] <http://www.rethinkrobotics.com/baxter/> [01/2017]

[26] *Baxter Tech Specs.* [Online] <http://www.rethinkrobotics.com/baxter/tech-specs/> [01/2017]

[27] *Baxter Hardware specs.* [Online] <http://sdk.rethinkrobotics.com/wiki/Hardware_Specifications> [01/2017]

[28] *ROS.* [Online] <http://www.ros.org/> [01/2017]

[29] *Polhemus Fastrak.* [Online] <http://polhemus.com/motion-tracking/all-trackers/fastrak> [01/2017]

[30] *Fastrak specs.* [Online] <http://polhemus.com/_assets/img/FASTRAK_Brochure.pdf> [01/2017]

[31] *Flask.* [Online] <http://flask.pocoo.org/> [01/2017]

[32] *Baxter Inverse Kinematics.* [Online] <http://sdk.rethinkrobotics.com/wiki/IK_Service_Example> [01/2017]

[33] *Baxter Inverse Kinematics solver service.* [Online] <http://sdk.rethinkrobotics.com/wiki/API_Reference#Inverse_Kinematics_Solver_Service> [01/2017]

[34] *Baxter Inverse Kinematics solve position.* [Online] <http://api.rethinkrobotics.com/baxter_core_msgs/html/srv/SolvePositionIK.html> [01/2017]

[35] *Baxter play record.* [Online] <http://sdk.rethinkrobotics.com/wiki/Joint_Trajectory_Playback_Example> [01/2017]